Examples of AT commands on STM32WB Series microcontrollers

Introduction

The STM32WB Series microcontroller is a very low-power Bluetooth® Low Energy (BLE) dual processor, compliant with Bluetooth Core Specification v5. The entire BLE stack runs on the embedded Arm® Cortex®-M0 core. The wireless processor interfaces with the Arm® Cortex®-M4 application microcontroller using the IPCC transport layer and a set of APIs composed of standard host controller interface (HCI) instructions and vendor-specific application command interface (ACI) instructions.

This application note describes the set of AT (attention) commands for a multi-app AT project implemented on STM32WB Series and explains the concepts of pairing and low-power used for this project.

This document explains how to interface with the STM32WB Series microcontroller to manage multi BLE apps handling (AT P2P server, AT P2P client and AT heart rate) using AT instructions.

1 General information

This document applies to the STM32WB Series dual-core Arm®-based Series microcontroller.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Overview

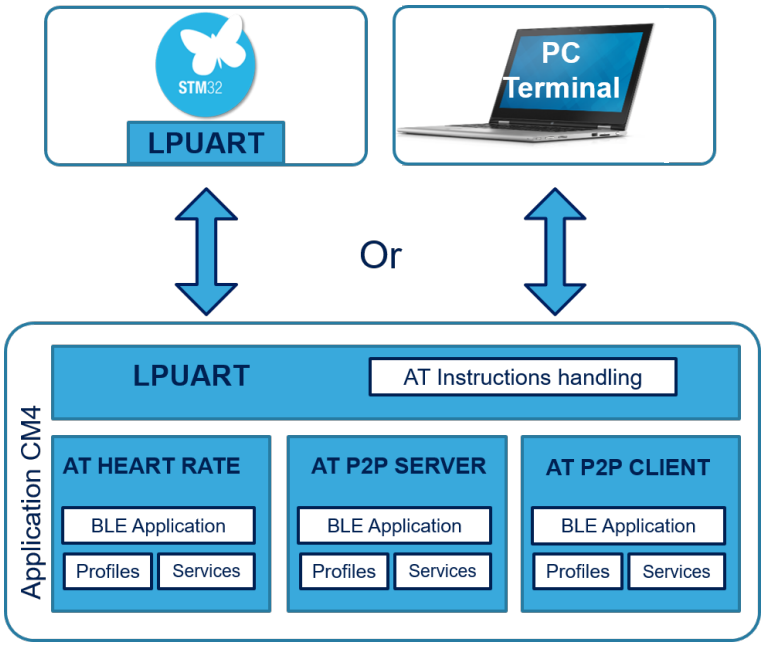
The multi app AT firmware applies to STM32WB Series microcontrollers, based on Arm® cores. It also embeds the firmware on the Arm® Cortex®-M4.

This project is used in a network processor architecture where a third party processor communicates via UART with an STM32WB Series microcontroller in order to use different apps and switch between each of them at any time.

This firmware implements the AT slave module that supports a set of AT instructions to drive the STM32WB Series microcontroller and uses the AT P2P server, AT P2P client and AT heart rate. These applications are respectively based on the P2P server app, P2P client app and heart rate app. Refer to *Building wireless applications with STM32WB Series microcontrollers* (AN5289) for details.

Section 3 contains the interface description and the AT instruction definitions. Section 4 the description of some use cases of the embedded software.

Figure 1. Multi app AT instruction Architecture



3 AT instructions

The AT instruction set is a standard developed by Hayes to control modems. AT stands for attention.

The instruction set consists of a series of short text strings providing operations such as data exchange and parameter settings.

In the context of a multi app AT project, the Hayes command set is a variation of the standard AT Hayes commands.

The AT instructions are used to drive the STM32WB Series microcontroller and to send and receive data. The AT instructions are sent through the UART and more precisely through the low-power UART (LPUART).

The STM32WB Series microcontroller can be controlled either through a terminal emulator such as Tera Term or PuTTY, or through an embedded AT master module.

The LPUART embedded on the STM32WB Series microcontroller is then used with standard Windows® based software as defined above: Tera Term or PuTTY. The chosen software must be configured with the following parameters:

- Baud rate: 9600
- Data: 8 bits
- Parity: none
- Stop: 1 bit
- Flow control: none.

Figure 2 and Figure 3 show the standard configuration for Tera Term to use the LPUART.

Figure 2. Tera Term terminal setup example

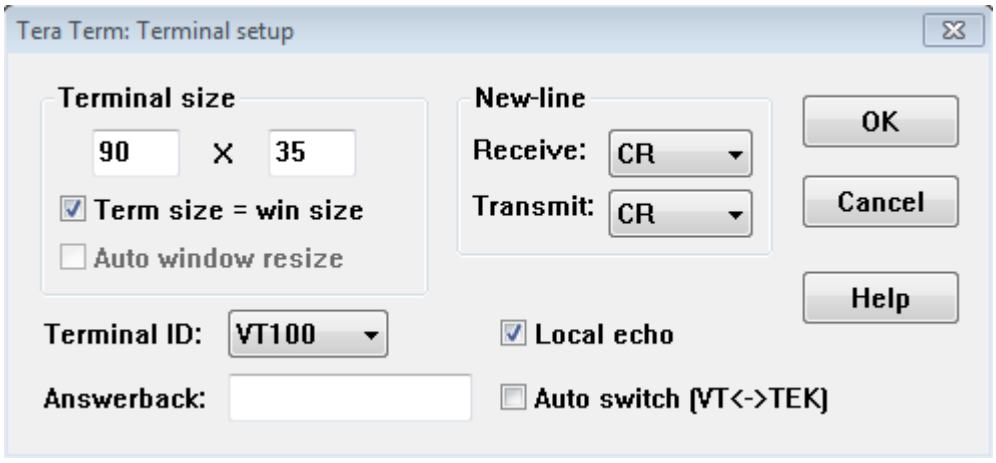
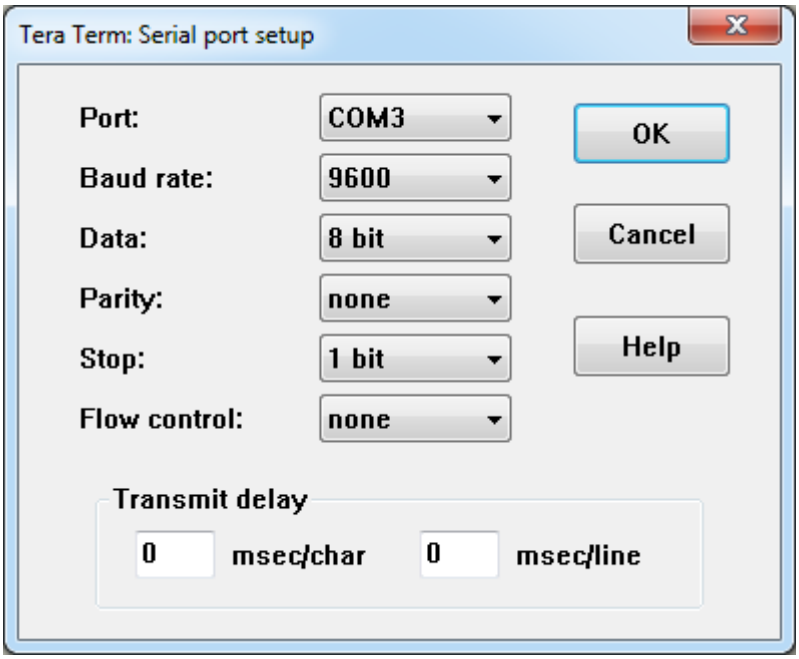


Figure 3. **Tera Term port setup example**



All the AT instructions have a standard format “AT+XXX”, with XXX denoting the instruction. There are three available instruction behaviors:

- AT+XX used to switch BLE app, such as AT+SV.
- AT+XX$XX used to run an instruction where first the app is specified and then the actual instruction is specified, such as AT+CL$SCAN (in client mode, I request a scan).
- AT+XX$XX=<value> used to provide a value to an instruction, for example AT+SV$NOTIFY=0001.

Each input instruction must have a <CR> at the end. This allows the module to know where the end of an instruction is:

AT+SV<CR>

The instruction output is provided on the LPUART. The output format is illustrated below:

```
<CR><LF><value><CR><LF>
```

```
<CR><LF><Status><CR><LF>
```

`<CR><LF><value><CR><LF>` is returned when `AT+CL$SCAN<CR>` is sent.

Otherwise, `<CR><LF><Status><CR><LF>` is returned.

Note: *`<CR>` stands for “carriage return” (usually `\r`) and `<LF>` stands for “line feed” (usually “`\n`”).*

Every instruction returns at least one status string, which is preceded and followed by `<CR><LF>` in a “`<CR><LF><Status><CR><LF>`” format. The possible statutes are:

- OK - Instruction run correctly without error
- ERROR - Generic error
- F - Connection/disconnection failure
- C - Connected
- D - Disconnected
- A - Advertising
- N = X {0, 1} - Notifications disabled/enabled
- NF - Not found when running an AUTOCONN instruction.

More details on each instruction description and examples are described in the next part of this section.

3.1 General instructions

This section describes the instructions related to “attention” instruction, and boots on different apps.

3.1.1 AT: attention

This instruction is used to check that the link is working properly (see table below).

Table 1. Check instruction

Instruction	Input parameter	Return value	Return code
AT	-	-	OK

3.1.2 AT+SV: boot on AT P2P server

The user boots on AT P2P server (see table below).

Table 2. Boot on AT P2P server instruction

Instruction	Input parameter	Return value	Return code
AT+SV	-	-	SV OK/ERROR

ERROR is returned when the current mode is already AT P2P server.

3.1.3 AT+CL: boot on AT P2P client

The user boots on AT P2P Client (see table below).

Table 3. Boot on AT P2P client instruction

Instruction	Input parameter	Return value	Return code
AT+CL	-	-	CL OK/ERROR

ERROR is returned when the current mode is already AT P2P client.

3.1.4 AT+HR: boot on AT heart rate

The user boots on AT heart rate (see table below).

Table 4. Boot on AT heart rate instruction

Instruction	Input parameter	Return value	Return code
AT+HR	-	-	HR OK/ERROR

ERROR is returned when the current mode is already AT heart rate.

3.2 AT P2P server specific instructions

This section gives the description of the instructions related to AT P2P server app.

3.2.1 AT+SV$ADV_START: start advertising

The user starts advertising (see table below).

Table 5. Start advertising instruction

Instruction	Input parameter	Return value	Return code
AT+SV$ADV_START	-	-	A/F/ERROR

ERROR is returned when the current mode is not AT P2P server.
F is returned when the advertising fails to start.

3.2.2 AT+SV$ADV_STOP: stop advertising

The user stops advertising (see table below).

Table 6. Stop advertising instruction

Instruction	Input parameter	Return value	Return code
AT+SV$ADV_STOP	-	-	OK/F/ERROR

ERROR is returned when the current mode is not AT P2P server.
F is returned when the advertising fails to stop.

3.2.3 AT+SV$NOTIFY: notify the client

The user notifies the client of the input data (see table below).

Table 7. Notify client instruction (AT P2P Server)

Instruction	Input parameter	Return value	Return code
AT+SV$NOTIFY=<input>	<2 bytes hex values>	-	OK/ERROR
Example: AT+SV$NOTIFY=	0001	-	OK

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT P2P server.

3.2.4 AT+SV$CONN_UPD: request to change connection interval

The user requests a connection interval change from 8 ms to 3 s for example (see table below).

Table 8. Connection update instruction

Instruction	Input parameter	Return value	Return code
AT+SV$CONN_UPD=<input>	[8;3000][ms]	-	OK/ERROR
Example: AT+SV$CONN_UPD=	500	-	OK

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT P2P server.

3.3 AT P2P client specific instructions

This section gives a description of the instructions related to AT P2P client app.

3.3.1 AT+CL$WRITE: send a write instruction

The user sends a write instruction (see table below).

Table 9. Write instruction

Instruction	Input parameter	Return value	Return code
AT+CL$WRITE=<input>	<2 hex values>	-	OK/ERROR
Example:AT+CL$WRITE=	0001	-	OK

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT P2P client.

3.3.2 AT+CL$EN: enable notifications

The user enables the notifications (see table below).

Table 10. Enable notifications instruction

Instruction	Input parameter	Return value	Return code
AT+CL$EN	-	-	OK/ERROR

ERROR is returned when the current mode is not AT P2P client.

3.3.3 AT+CL$DIS: disable notifications

The user disables the notifications (see table below).

Table 11. Disable notifications instruction

Instruction	Input parameter	Return value	Return code
AT+CL$DIS	-	-	OK

ERROR is returned when the current mode is not AT P2P client.

3.3.4 AT+CL$SCAN: scan the BLE devices

The user scans the discoverable BLE devices nearby. It returns all the identified BD addresses that match the P2P server app (see table below).

Table 12. Scan BLE devises instruction

Instruction	Input parameter	Return value	Return code
AT+CL$SCAN	-	<BD addresses>	OK/F/ERROR

ERROR is returned when the current mode is not AT P2P client.

F is returned when the scan fails.

3.3.5 AT+CL$CONN: connect to device

The user connects to the device whose BD address is the input parameter (see table below).

Table 13. Connect to device instruction

Instruction	Input parameter	Return value	Return code
AT+CL$CONN=<input>	<6 bytes hex values>	-	C/F/ERROR
Example: AT+CL$CONN=	74DA0025E180	-	C

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT P2P client.

F is returned when the connection request fails.

3.3.6 AT+CL$DISCONN: disconnect from device

The user disconnects the client from the server (see table below).

Table 14. Disconnect from device instruction

Instruction	Input parameter	Return value	Return code
AT+CL$DISCONN	-	-	D/F/ERROR

ERROR is returned when the current mode is not AT P2P client.

F is returned when the disconnection request fails.

3.3.7 AT+CL$AUTOCONN: auto-connection to device

The user connects to the device whose BD address is the input parameter without performing a scan beforehand as the scan is included in the instruction (see table below).

Table 15. Auto-connection to device instruction

Instruction	Input parameter	Return value	Return code
AT+CL$AUTOCONN=<input>	<6 hex values>	-	C/F/NF/ERROR
Example: AT+CL$AUTOCONN=	74DA0025E180	-	C

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT P2P client.

F is returned when the connection request or the scan fails.

NF is returned when the given BD address is not found at the end of the scan.

3.4 AT heart rate specific instructions

This section describes the instructions related to AT heart rate app.

3.4.1 AT+HR$NOTIFY: notify the client

The user notifies the client with expended energy values (in kJ) as well as the heart rate in beats per minute (see table below).

Table 16. Notify client instruction (AT heart rate)

Instruction	Input parameter	Return value	Return code
AT+HR$NOTIFY=<Heart Rate>,<Energy Expended>	Integer	-	OK/ERROR
Example: AT+HR$NOTIFY=	70,120	-	OK

ERROR is returned when:

- Setting a wrong or malformed value.
- The current mode is not AT heart rate.

3.5 Instructions specific to pairing

This section describes of the instructions related to the pairing process. For more information about the pairing process, refer to Section 5.3 Pairing process.

3.5.1 AT+S$PAIRING_START: start pairing

The user starts a pairing procedure (see table below).

Table 17. Start pairing instruction

Instruction	Input parameter	Return value	Return code
AT+S$PAIRING_START	-	-	-

When phase I of the pairing has finished correctly, meaning the master has sent a pairing request and the slave has sent a pairing response. Phase II starts and the user receives a confirmation request in the form of `<6 digits>: PREQ?`.

3.5.2 AT+S$PAIRING_CONFIRM: confirm pairing

This instruction lets the user confirm the pairing on the slave side after verifying the matching numbers on the terminal and the smartphone (see table below).

Table 18. Confirm pairing instruction

Instruction	Input parameter	Return value	Return code
AT+S$PAIRING_CONFIRM= <input>	Y or N	-	P/F/ERROR
Example: AT+S$PAIRING_CONFIRM =	Y	-	P

ERROR is returned when setting a wrong or malformed value.

F is returned when the pairing fails.

3.5.3 AT+S$CLEAN_BONDING: clean secure data

The user cleans the secure data in order to perform a complete pairing process on next connection (see table below).

Note: Cleaning the secure data on slave side is not sufficient; secure data on the master side has to be cleaned also, meaning forgetting the device on the smartphone.

Table 19. Clean secure data instruction

Instruction	Input parameter	Return value	Return code
AT+S$CLEAN_BONDING	-	-	OK

4 Examples

This section provides some example of AT P2P server execution, AT P2P client, AT heart rate and a common switching app pattern.

4.1 AT P2P server

This example shows one possible use case when the current mode is P2P server.

```
#  AT<CR>
<CR><LF>OK<CR><LF>
#  AT+SV<CR>
<CR><LF>SV OK<CR><LF>
<CR><LF>A<CR><LF> /* Advertising */
/* Connecting the server to a smartphone */
<CR><LF>C<CR><LF> /* Connected */
/* As a client/smartphone enables the notifications */
<CR><LF>N=1<CR><LF>
#  AT+SV$NOTIFY=0001<CR>
<CR><LF>OK<CR><LF> /* In the current app, a text appears on the smartphone screen */
/* As a client, a write is sent to the corresponding characteristic of the server */
<CR><LF>R=0101<CR><LF> /* In the current code, this turns on the blue led */
<CR><LF>R=0100<CR><LF> /* In the current code, this turns off the blue led */
<CR><LF>R=0101<CR><LF>
#  AT+SV$CONN_UPD=500<CR>
<CR><LF>OK<CR><LF>
```

4.2 AT P2P client

This example shows one possible use case when the current mode is P2P client.

```
#  AT<CR>
<CR><LF>OK<CR><LF>
#  AT+CL<CR>
<CR><LF>CL OK<CR><LF>
#  AT+CL$SCAN<CR>
<CR><LF>74DA0025E180<CR><LF>
<CR><LF>42E50025E180<CR><LF>
<CR><LF>OK<CR><LF>
#  AT+CL$CONN=74DA0025E180<CR>
<CR><LF>C<CR><LF> /* Connected */
<CR><LF>OK<CR><LF>
#  AT+CL$EN<CR>
<CR><LF>N=1<CR><LF>
#  AT+CL$WRITE=0001<CR>
<CR><LF>OK<CR><LF> /* This turns on the blue led on server side */
#  AT+CL$DISCONN<CR>
<CR><LF>D<CR><LF>
<CR><LF>OK<CR><LF>
```

4.3 AT heart rate

This example shows one possible use case when the current mode is heart rate.

```
#  AT<CR>
<CR><LF>OK<CR><LF>
#  AT+HR<CR>
<CR><LF>HR OK<CR><LF>
<CR><LF>A<CR><LF> /* Advertising */
/* Connecting the server to a smartphone */
<CR><LF>C<CR><LF> /* Connected */
/* As a client/smartphone, the notifications are enable */
<CR><LF>N=1<CR><LF>
#  AT+HR$NOTIFY=70,120<CR>
<CR><LF>OK<CR><LF>
/* On the client side, the energy expended is now 120 kJ and the heart rate is 70 bpm */
```

4.4 Switching app

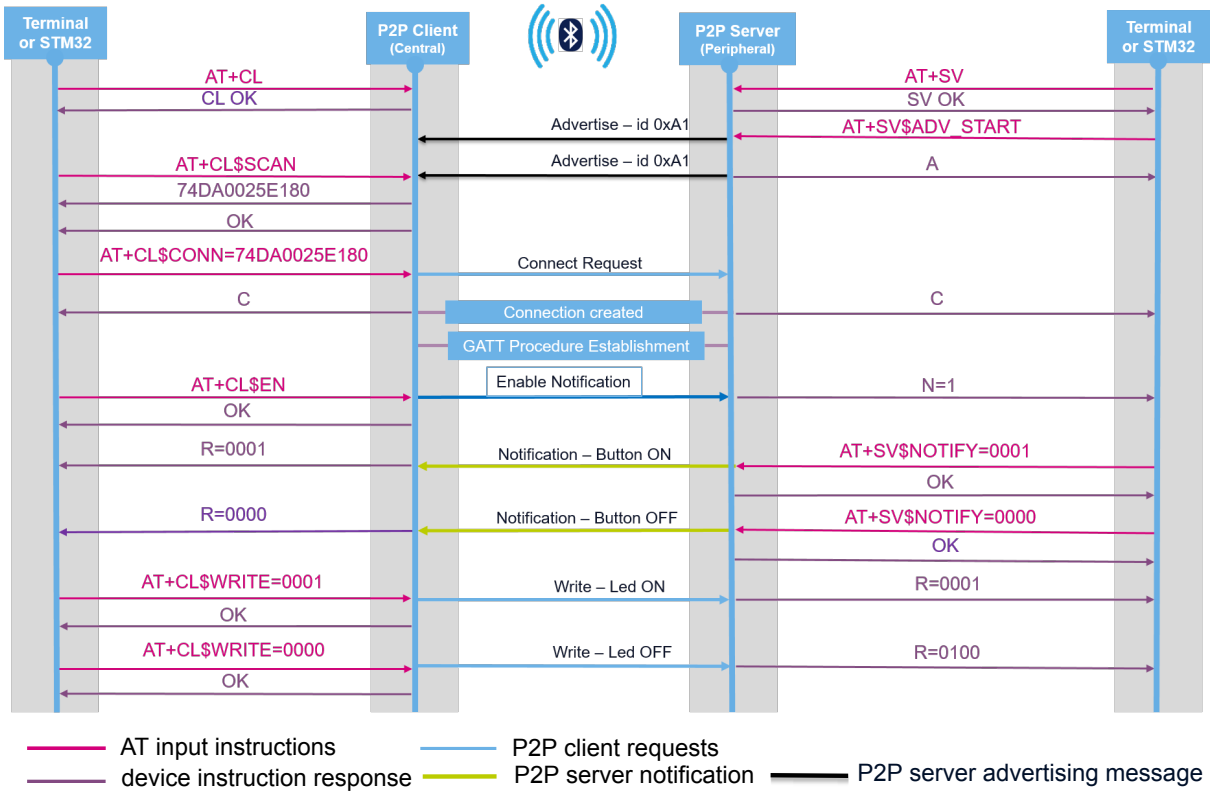
This example shows one possible use case when you switch app.

```
#  AT<CR>
<CR><LF>OK<CR><LF>
#  AT+SV<CR>
<CR><LF>SV OK<CR><LF> /* Server mode */
#  AT+HR<CR>
<CR><LF>HR OK<CR><LF> /* Heart Rate mode */
#  AT+CL<CR>
<CR><LF>CL OK<CR><LF> /* Client mode */
```

4.5 Sequence diagram

The diagram below illustrates the sequencing of operations between an AT P2P server and an AT P2P client.

Figure 4. Sequence diagram between an AT P2P server and AT P2P client



5 Embedded software description

This section gives an overview of the firmware architecture STM32WB55RG

5.1 Firmware overview

The AT instruction processing is found in the source files listed below:

- `uart_app.c`: contains AT driver functions
- `uart_app.h`: contains the definition for all the instructions.

5.2 LPUART

The AT instructions are sent through a UART carrier. In order to optimize the use of low power, the LPUART of the STM32WB board is used.

The AT slave module executes two different tasks:

- BLE tasks: manages the connection, data transfer (write, notification), and so on.
- Receives instructions from the master that dictates its behavior, and then returns the requested instruction value and status.

As a non-blocking UART communication is used, the slave is listening all the time but as it receives on interrupt (IT), this does not consume much power.

The MCU constantly waits for an instruction from the master and therefore remains in idle state most of the time.

Therefore, it is important to be in Stop mode in order to optimize the MCU's low-level power capability. As instructions are received through the UART and directed to the low-power UART (LPUART).

The `HW_UART_Receive_IT()` function enables RXNE (RX not empty) IT, so that when RXNE IT is raised, a custom callback `RxCpltCallback()` is called where the character in the register is read and stored in a buffer.

When a carriage return is detected as a character, the buffer of read characters is then processed in a normal thread (not in the IT thread).

5.3 Pairing process

When two connected devices want to start a secure operation, they initiate a specific pairing process. This procedure is triggered by a master (such as a smartphone). This pairing is usually necessary when the master/client wants to access a slave/server service that requires authentication. To put it simply, a pairing procedure starts by authenticating the identity of the two devices, encrypting the link using a short-term key (STK) and then distributing a long-term key (LTK) used for data encryption. The LTK is saved for a faster reconnection in the future. The devices are then said to be bonded.

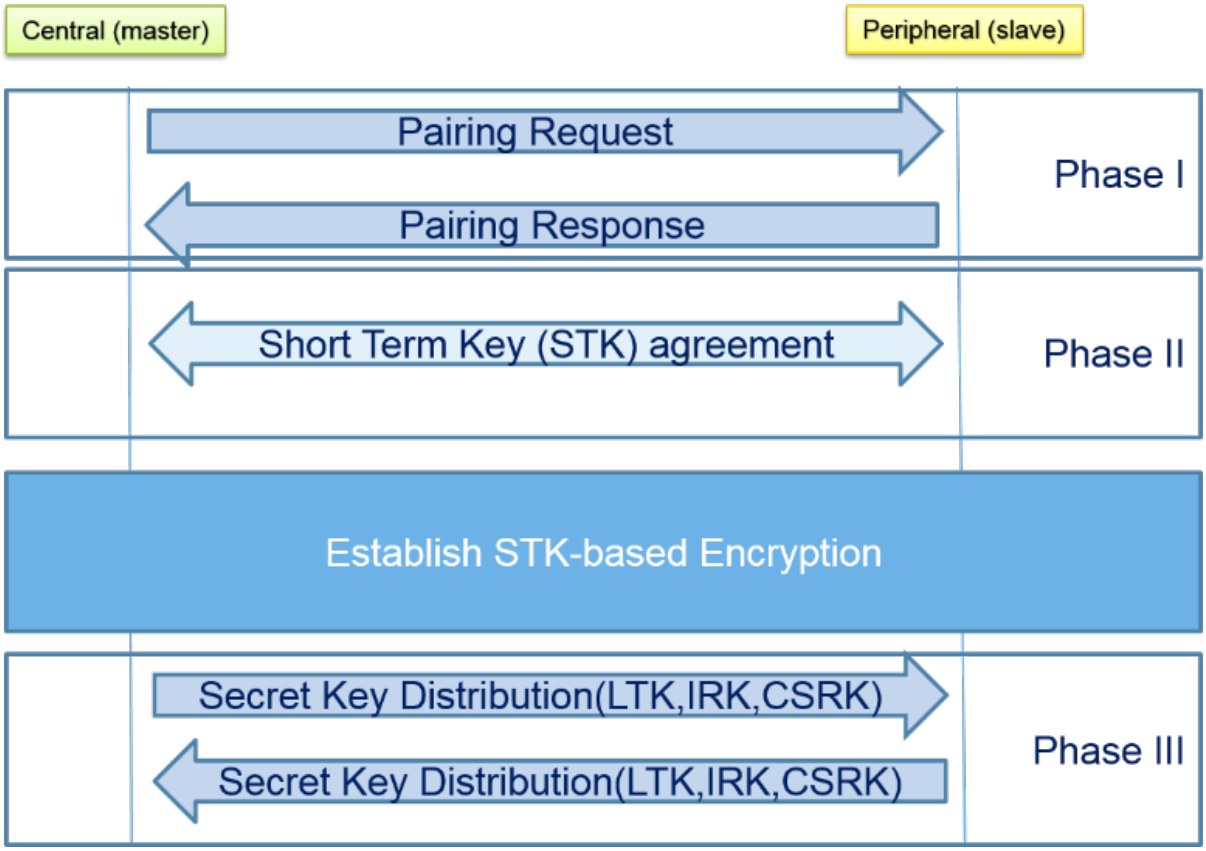
The encryption in Bluetooth® LE is based on *128-bit Advanced Encryption Standard—Counter with CBC-MAC (AES-CCM)*. LTK is used with this algorithm to create the 128-bit "shared secret" key.

BLE describes two security models for pairing that is be explained in the next sections:

- LE legacy connections (Section 5.3.1)
- LE secure connections (Section 5.3.2).

5.3.1 LE legacy connections

Figure 5. **LE legacy connections overview**

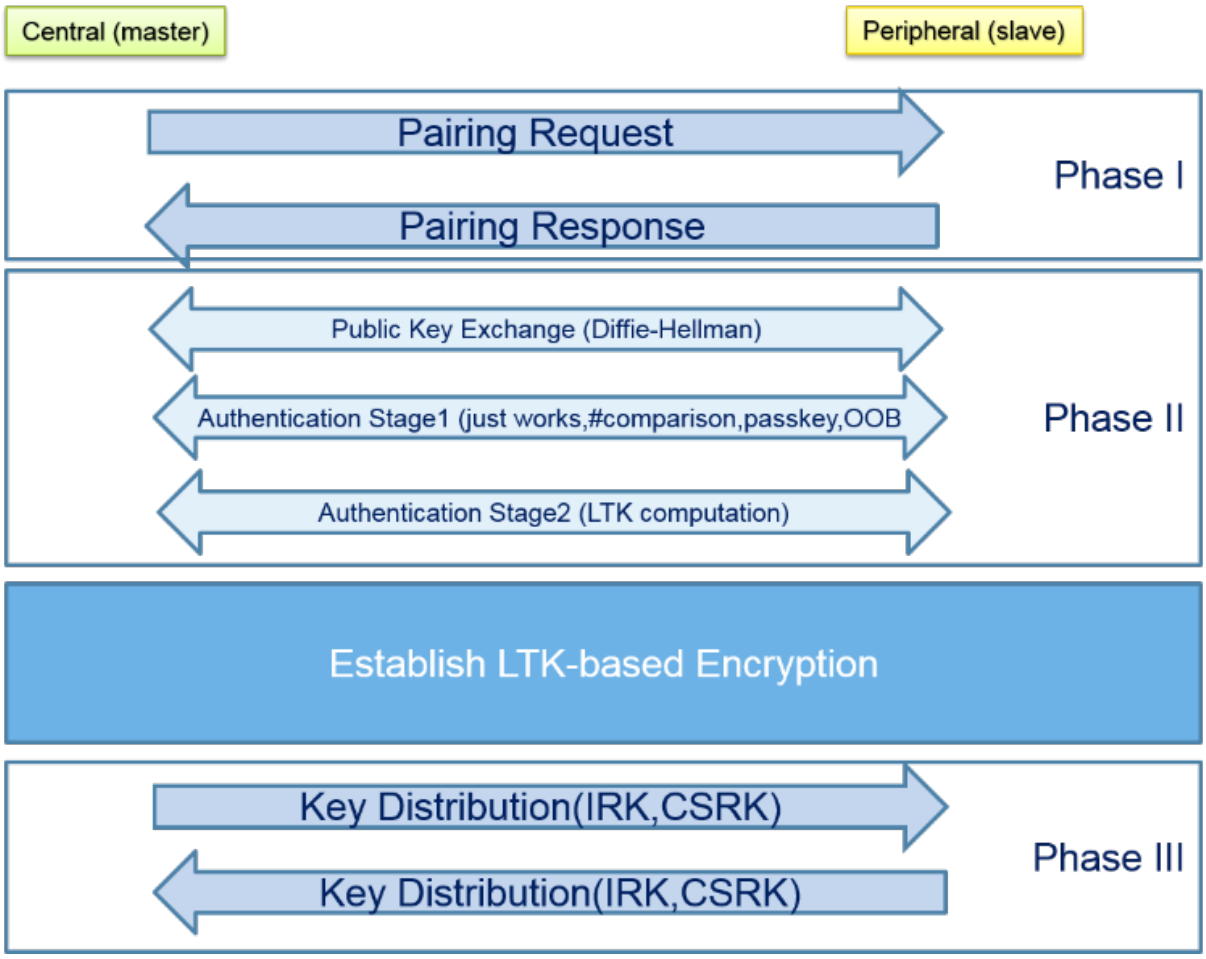


In this model (see Figure 5), the devices exchange a temporary key (TK) which is used to create a short term key (STK). The STK is then used to encrypt the connection. The way the TK is exchanged greatly influences the connection security. Three methods can be applied to exchange the temporary key:

- Just works: In this method the TK is set to 0, therefore the STK generated on both sides can easily be broken if the encryption algorithm is known. Furthermore there is no protection against man-in-the-middle (MITM) attacks.
- Out of band (OOB): In this method, the TK is exchanged using another wireless technology such as NFC. If the wireless technology used provides protection against MITM attacks, then it is reasonable to expect that the BLE connection and the OOB pairing are also immune to MITM attacks.
- Passkey: In this method, the TK is a 6 digits number that is passed between the devices by the user. Which means, one device displays its TK and the user enters the number into the second device using a keypad for instance. Subsequently, both devices have the same TK and can generate the same STK. In some cases, the user needs to enter the same key on both sides if no display is available. This method provides a good protection from MITM attacks as well as passive eavesdropping, assuming that no attacker is listening during the pairing process.

5.3.2 LE secure connections

Figure 6. **LE Secure Connections overview**



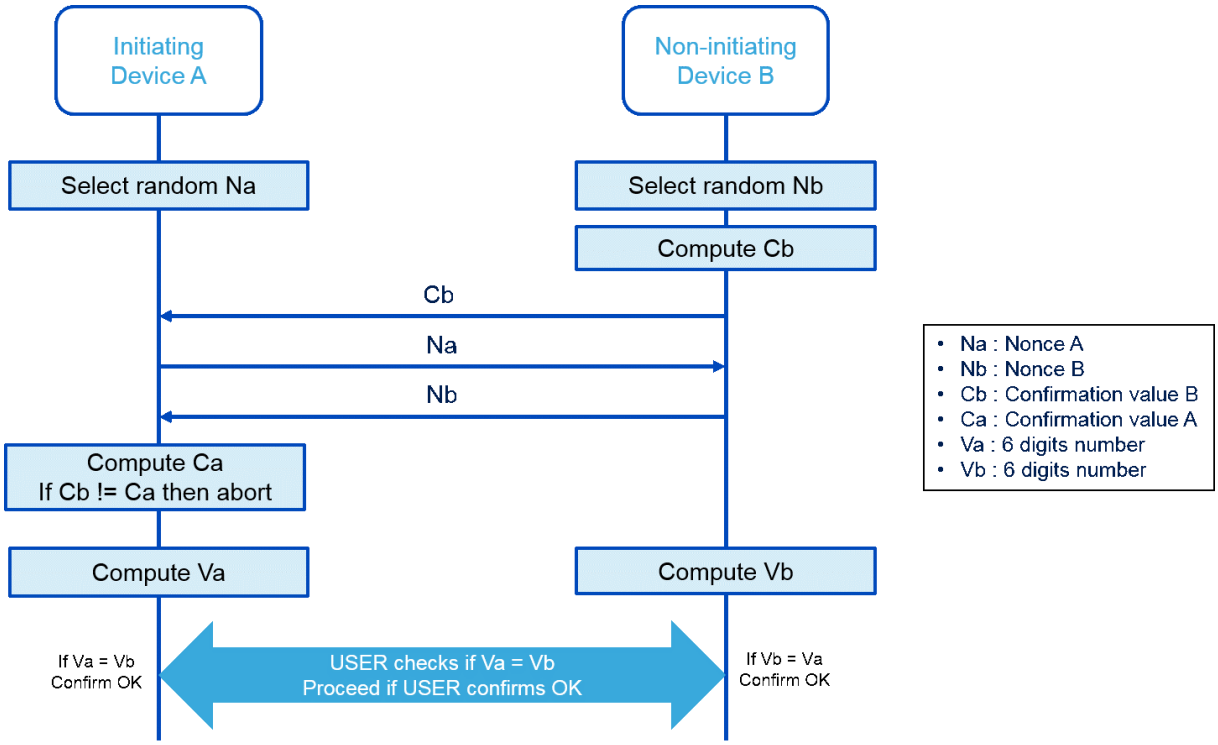
In this model (illustrated in Figure 6), instead of using a TK and STK, a single long term key (LTK) is used to encrypt the link. At the beginning of phase II, the devices exchange their public key, and then the non-initiating device (in this case the slave) generates a nonce (arbitrary number that can be used just once in a cryptographic communication) that generates a confirmation value Cb. The Cb is sent to the initiating device (in this case the master). Concurrently, the initiating device generates its own nonce and sends it to the non-initiating device. The initiating device then uses the non-initiating device's nonce to generate its own confirmation value Ca which must match Cb. If the confirmation values match, then the connection proceeds. This procedure gives significant protection against passive eavesdropping compared to its legacy counterpart.

Four methods can be applied to encrypt the connection in LE secure connections model of which three of them are similar to the legacy model. However, as seen above, in general, the LTK exchange in the secure model ensures a better protection from passive eavesdropping:

- Just works: This method is described above and there is no action from the user.
- Out of band (OOB): This method is the same as in the legacy model instead of exchanging the TK via a different wireless technology such as NFC, the public keys, nonces and confirmation values are exchanged.
- Passkey: This method follows the same principle as its legacy counterpart. An identical 6 digits number is input into both devices. These two devices then use this passkey as well as a public key and a nonce to authenticate the connection.
- Numeric comparison: This pairing method uses the same procedure as the just works pairing method but adds another step which involves the user and provides a protection from MITM attacks. Once both confirmation values match, a final 6 digits number is generated independently on both devices and displayed. Then the user manually checks if both numbers are the same and confirms the pairing.

Figure 7 illustrates a sequence diagram of the secure connections model using the numerical comparison method.

Figure 7. Secure connections model - numeric comparison method



5.3.3 Security modes

In the BLE protocol, the generic access profile (GAP) defines two security modes along with several security levels. In this section, only the security mode 1 is considered as the second one is not implemented in the STM32WB Series microcontrollers' BLE stack.

The security mode 1 ensures security by means of encryption and contains four levels:

- Security level 1: No security (by default when two devices are connected)
- Security level 2: Unauthenticated pairing with encryption
- Security level 3: Authenticated pairing with AES-CCM encryption
- Security level 4: Authenticated LE secure connection pairing with encryption. Level 4 uses Elliptic Curve Diffie-Hellman P-256 (ECDH) and AES-CCM encryption.

5.3.4 Project pairing settings

For pairing, the LE secure connections with numerical comparison is used. In other words, the security mode 1 Level 4. Nevertheless, in case an old device that does not support BLE 4.2 is used, pairing with the LE legacy connections is used. However, it is necessary to understand the potential security flaws when using LE legacy connections.

The table below shows a comprehensive description of the pairing settings.

Table 20. Pairing settings

Settings	Values
IO capabilities	Display yes / no
Minimum encryption key size	8
Maximum encryption key size	16
Use fixed pin	1 (no fixed pin used)
Bonding mode	1 (the bonding information is stored in flash, therefore no need to redo the process on new connection)
MITM mode	1 (MITM protection required)
Secure support	1 (secure connections pairing supported but optional)
Keypress notification support	0 keypress notification not supported

5.4 Low power

To take advantage of the low power while keeping the LPUART running, the low speed external (LSE) clock running at 32.768 kHz has to be used, which limits the data rate to 9600 bauds.

For more details about low-power features, refer to *STM32WB ultra-low-power features overview* (AN5071) available on www.st.com.

Revision history

Table 21. **Document revision history**

Date	Version	Changes
24-Sep-2019	1	Initial release.
10-Feb-2020	2	Updated: • Table 7. Notify client instruction (AT P2P Server) • Table 13. Connect to device instruction

Contents

List of figures

List of tables