Developer Study Guide

Using BlueZ as a Bluetooth® Mesh Provisioner

Release : 1.4.1

Document Version: 1.4.1

Last updated : 18th December 2020

Contents

Contents

Revision History

Version	Date	Author	Comments
1.0	18th June 2018	Kai Ren Bluetooth SIG	Initial Draft
1.1	5th August 2018	Kai Ren Bluetooth SIG	Upgrade BlueZ installation to v5.50.
1.2	9th March 2019	Kai Ren Bluetooth SIG	Updated the name to Developer Study Guide. Use the latest Raspberry Pi release instead of main tree.
1.3	26th July 2019	Kai Ren Bluetooth SIG	Add the support for **Raspberry Pi 4** and update the kernel to raspberrypi-kernel_1.20190709-1.
1.4	16th March 2020	Kai Ren Bluetooth SIG	Upgrade this guide to support BlueZ v5.54 which adds the new support for PB-ADV
1.4.1	18th December 2020	Martin Woolley Bluetooth SIG	Language changes

1. Introduction

BlueZ is the official Linux *Bluetooth*® protocol stack. From the release notes of BlueZ v5.47:

> *"This release comes with initial support for it in the form of a new meshctl tool. Using this tool, it's possible to provision mesh devices through the GATT Provisioning Bearer (PB-GATT), as well as communicate with them (e.g. configure them) using the GATT Proxy protocol."*

This developer study guide explains how to install the latest release of BlueZ on Raspberry Pi and use BlueZ as a Bluetooth mesh Provisioner.

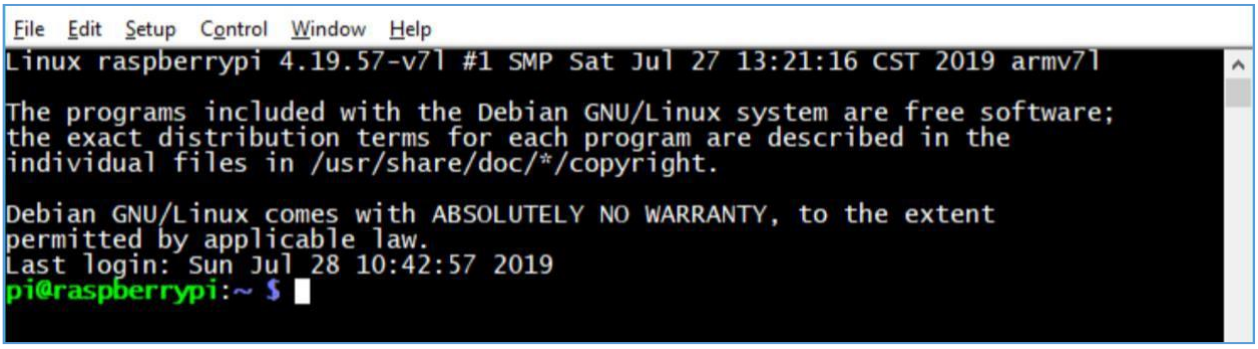
2. Prerequisite

This study guide has been tested on the following boards, calling them **verified boards** in this document:

- Raspberry Pi 2B
- Raspberry Pi 3B
- Raspberry Pi 3B+
- Raspberry Pi 4B

If you have one of above-verified boards, please make sure that you:

- Follow this guide to setup your Raspberry Pi
- Check if the operating system on your verified board is ready, and, if not, follow this guide to set up the software on your Raspberry Pi
- Follow this guide to enable SSH to access the board remotely. The picture below shows the use of Tera Term on a Windows10 laptop through SSH to access the board remotely



- The board has been issued *apt-get update* and *apt-get upgrade* successfully, these two commands will ensure your board has the latest updates

3. Rebuilding the Kernel for BlueZ

There are two main methods for building the Raspberry Pi kernel. You can build locally on a Raspberry Pi, which will take a long time, or you can cross-compile, which is much quicker but requires more setup. This guide outlines the local-build method.

3.1 Remote Access Board Through SSH

As mentioned in the Prerequisite, you should remote login into the board through SSH.

3.2 Install Dependencies for BlueZ

```
sudo apt-get install -y git bc libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev libical-dev
libreadline-dev autoconf bison flex libssl-dev
```

3.3 Check-out Source Code

```
cd ~
wget https://github.com/raspberrypi/linux/archive/raspberrypi-kernel_1.20200212-1.tar.gz
tar -xvf raspberrypi-kernel_1.20200212-1.tar.gz
```

3.4 Configuring the kernel

```
cd ~
cd ./linux-raspberrypi-kernel_1.20200212-1/
```

Depending on Raspberry Pi board models, run the following commands alternatively.

- Raspberry Pi 2, Pi 3, Pi 3+, and Compute Module 3[1] default build configuration

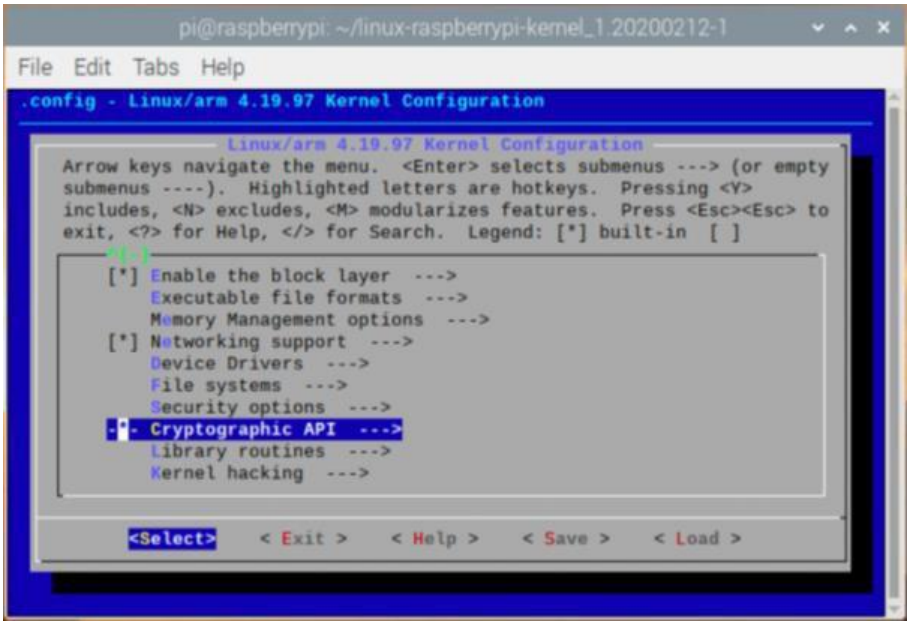
```
KERNEL=kernel7
make bcm2709_defconfig
make menuconfig
```

- Raspberry Pi 4

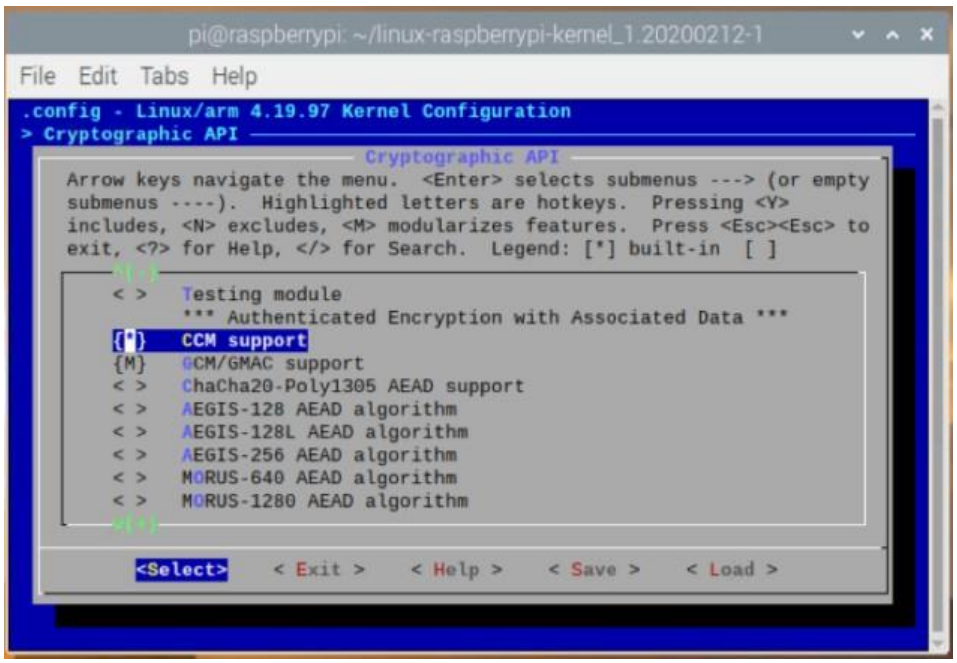
```
KERNEL=kernel7l
make bcm2711_defconfig
make menuconfig
```

After typing *menuconfig*, kernel configuration menu will pop up. *make menuconfig* shows the descriptions of each feature, gives the user an ability to navigate forwards or backwards directly between features and adds some dependency checking.
Route and select *Cryptographic API* menu:

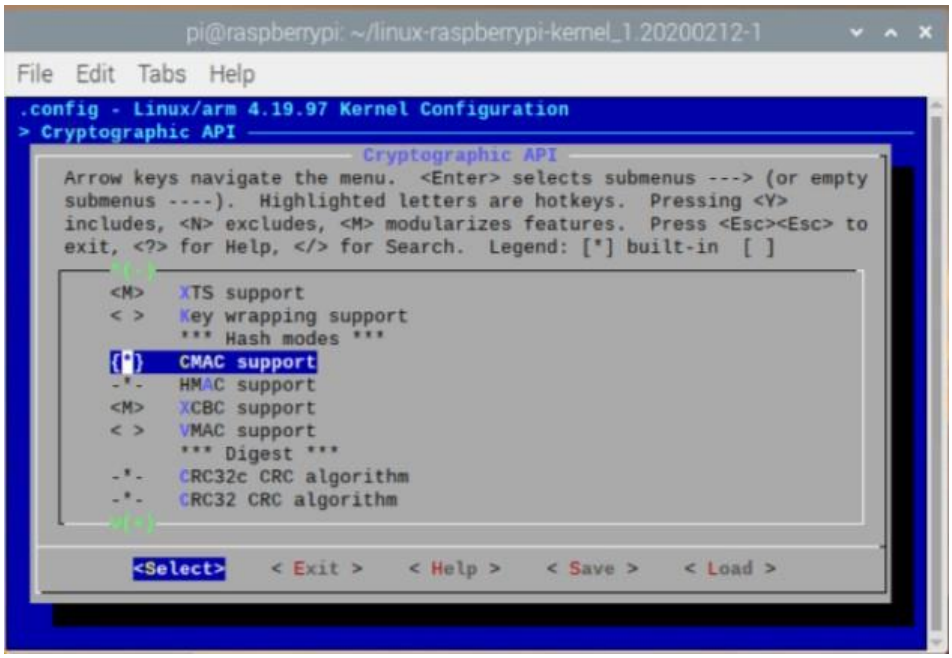
[1] Compute Module 3 haven't been verified on this document, but theoretically, it shall works.



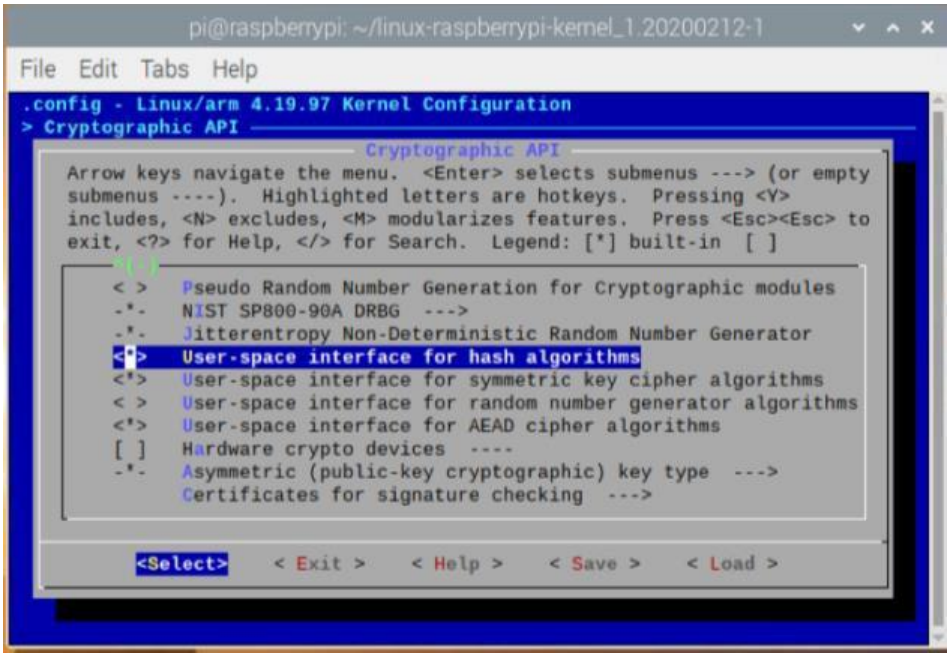
Include *CCM support*



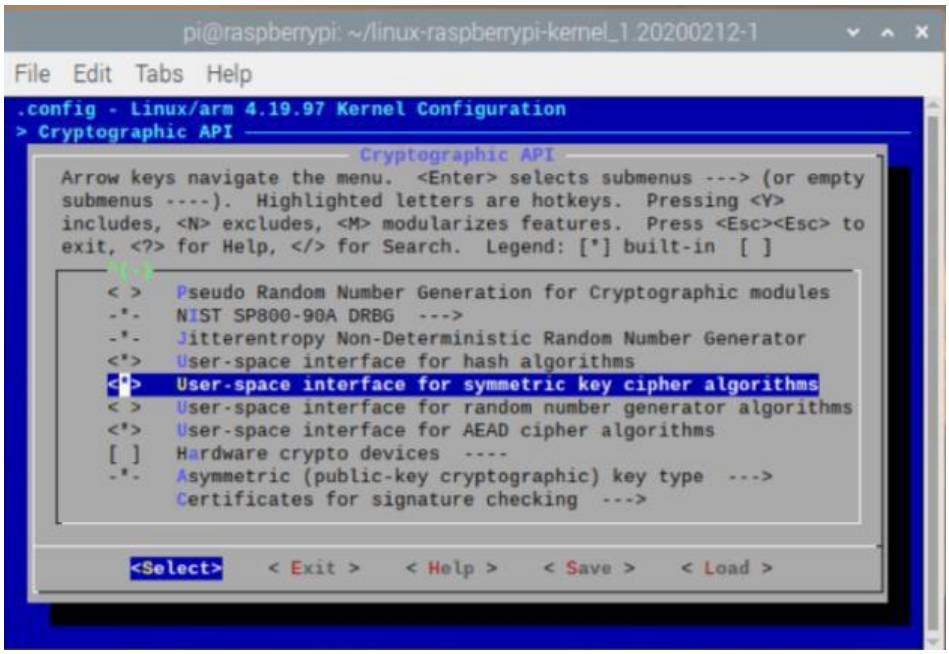
Include *CMAC support*



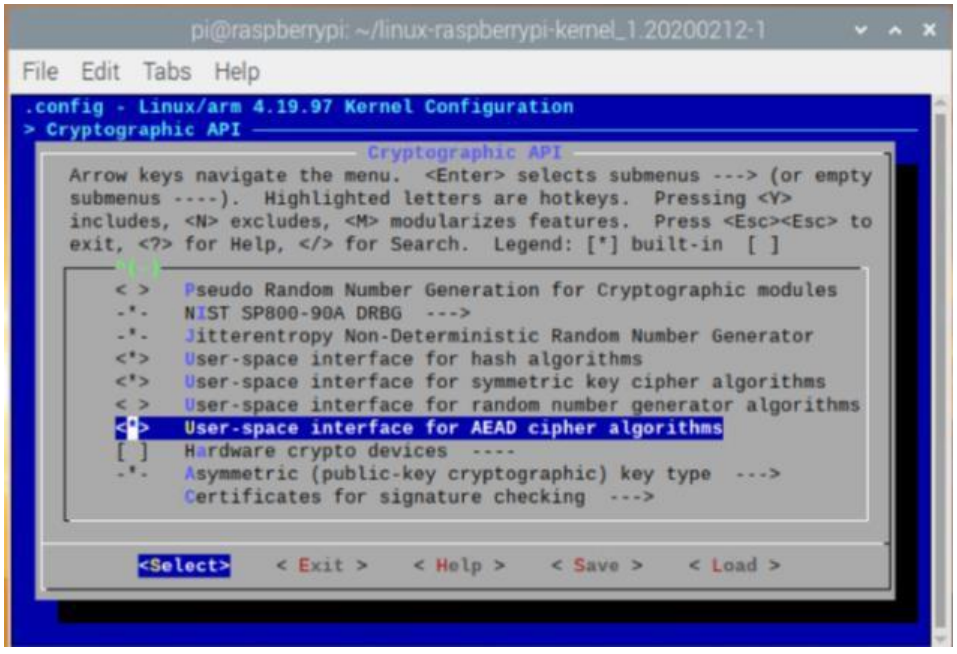
Include *User-space interface for hash algorithms*



Include *User-space interface for symmetric key cipher algorithms*



Include *User-space interface for AEAD cipher algorithms*



Once you are done making the changes you want, press *Escape* until you're prompted to save your new configuration. By default, this will save to *.config* file. You can save and load configurations by copying this file around.

3.5 Building the kernel

```
make -j4 zImage modules dtbs
```

This process takes a long time, maybe 2 ~ 3 hours.

3.6 Installing the kernel, Modules, and Device Tree Blobs

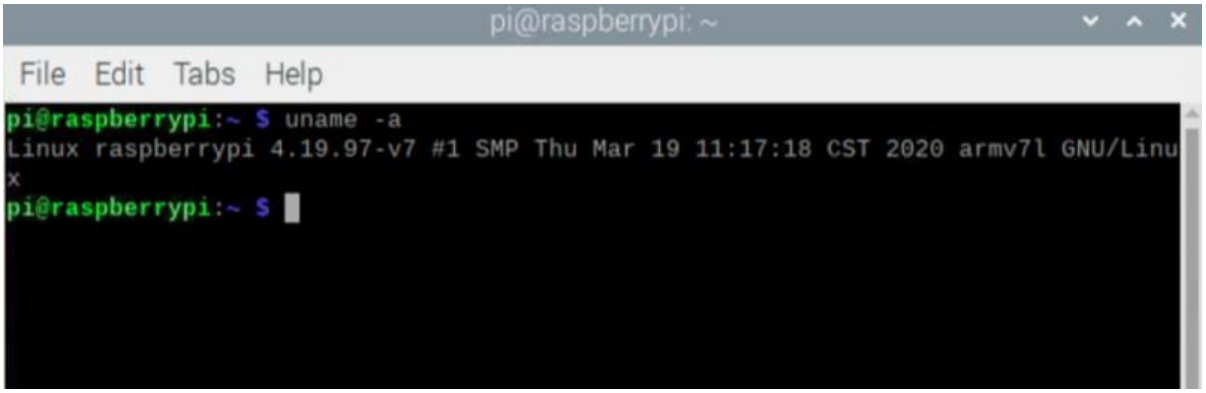
```
sudo make modules_install
sudo cp arch/arm/boot/dts/*.dtb /boot/
sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/overlays/
sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
sudo cp arch/arm/boot/zImage /boot/$KERNEL.img
sudo reboot
```

3.7 Verifying the kernel

After the board restart, issue below command

```
uname -a
```

In the image below, you can see the build time is on **Thu Mar 19 11:17:18 CST 2020**. That time and date were exactly when the kernel was built and it means the kernel building and installation were successful.



4. Installing BlueZ

Once the recompiled kernel is setup correctly, you can start to install BlueZ.

4.1 json-c installation

```
cd ~
wget https://s3.amazonaws.com/json-c_releases/releases/json-c-0.13.tar.gz
tar -xvf json-c-0.13.tar.gz
cd json-c-0.13/
./configure --prefix=/usr --disable-static && make
sudo make install
```

4.2 Get BlueZ Source Code

```
cd ~
wget http://www.kernel.org/pub/linux/bluetooth/bluez-5.54.tar.xz
tar -xvf bluez-5.54.tar.xz
cd bluez-5.54/
```

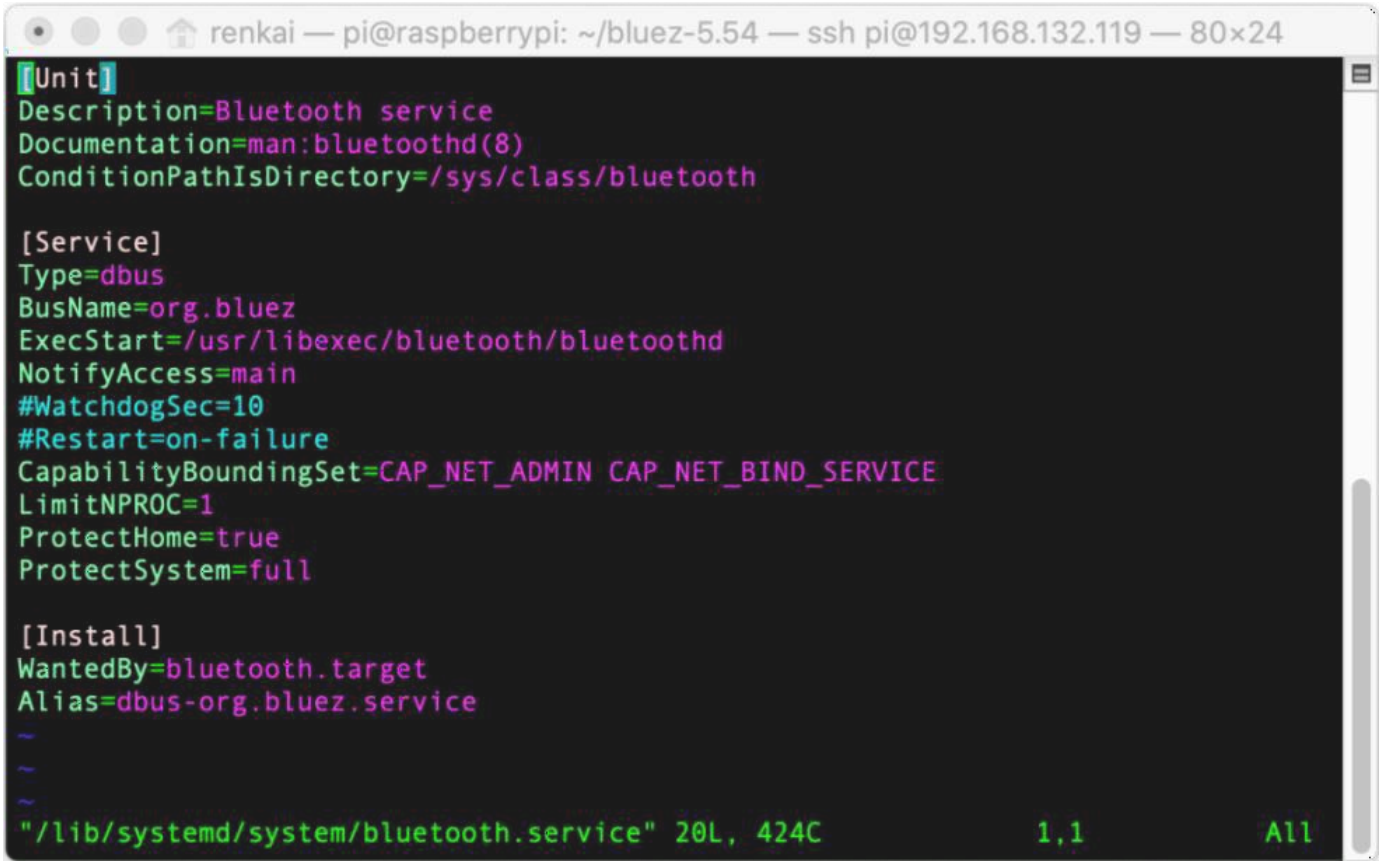
4.3 Build and Install BlueZ

```
./configure --enable-mesh --enable-testing --enable-tools --prefix=/usr --mandir=/usr/share/man --sysconfdir=/etc --localstatedir=/var
sudo make
sudo make install
```

4.4 Tell *systemd* to use the new Bluetooth daemon

```
sudo vi /lib/systemd/system/bluetooth.service
```

After opening this file, *bluetooth.service*, make sure the *ExecStart* line points to your new daemon in */usr/libexec/bluetooth/bluetoothd,* as shown in the screenshot below.



It's still not enough. You still need to create a symlink from the old *bluetoothd* to the new one. First, rename the old file for backup. Type below command.

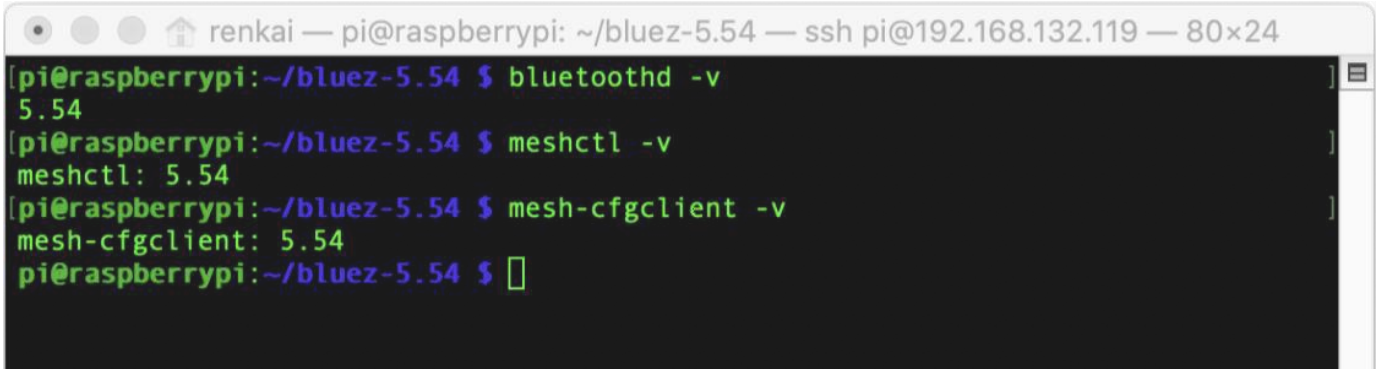
```
sudo cp /usr/lib/bluetooth/bluetoothd /usr/lib/bluetooth/bluetoothd-550.orig
```

Create the symlink using the command below and double check the version of *bluetoothd*, *meshctl* and *mesh-cfgclient*.

```
sudo ln -sf /usr/libexec/bluetooth/bluetoothd /usr/lib/bluetooth/bluetoothd
sudo systemctl daemon-reload
cd ~/.config/
mkdir meshctl
cp ~/bluez-5.54/tools/mesh-gatt/local_node.json ~/.config/meshctl/
cp ~/bluez-5.54/tools/mesh-gatt/prov_db.json ~/.config/meshctl/
bluetoothd -v
meshctl -v
mesh-cfgclient -v
```

As shown in the screenshot below, *bluetoothd*, *meshctl* and *mesh-cfgclient* are all v5.54.

Cheers! BlueZ installation is successful. [2]



[2] About upgrading *bluetoothd*, reference this article, https://raspberrypi.stackexchange.com/questions/66540/installing-bluez-5-44-onto-raspbian

5. Provisioning

This section includes the instructions that:

- how to use *meshctl* to provision a device over PB-GATT;
- how to use *mesh-cfgclient* to provision an unprovisioned device over PB-ADV;

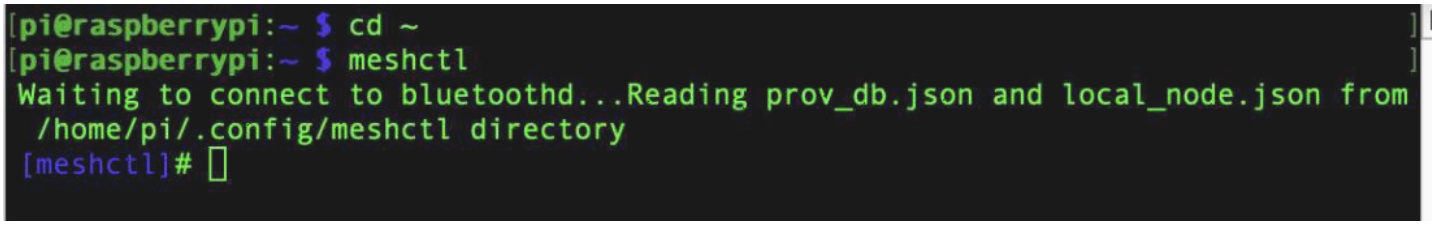
5.2 PB-GATT

meshctl is a tool that provides Provisioner functionality and it's over PB-GATT.

Launch meshctl

```
cd ~

meshctl
```

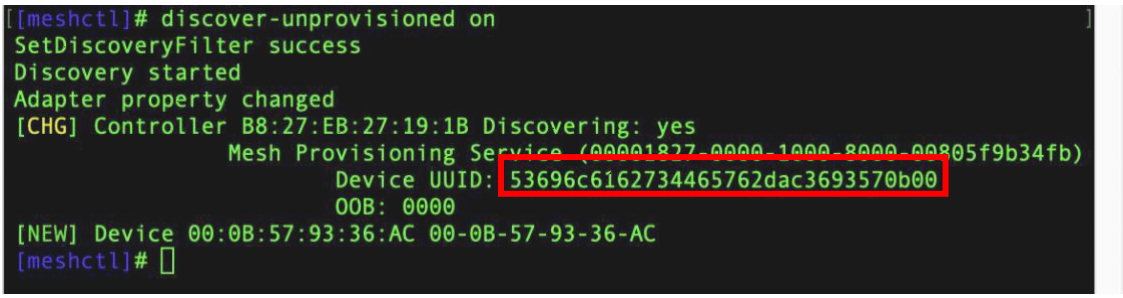


Discover unprovisioned devices

Start the process of discovering unprovisioned devices.

```
discover-unprovisioned <on/off>
```

<on/off> mean start or stop discovering unprovisioned device.
When an unprovisioned device is found, the message will pop-up as below the picture shown, the "Device UUID" in the red box will be used in the next section.



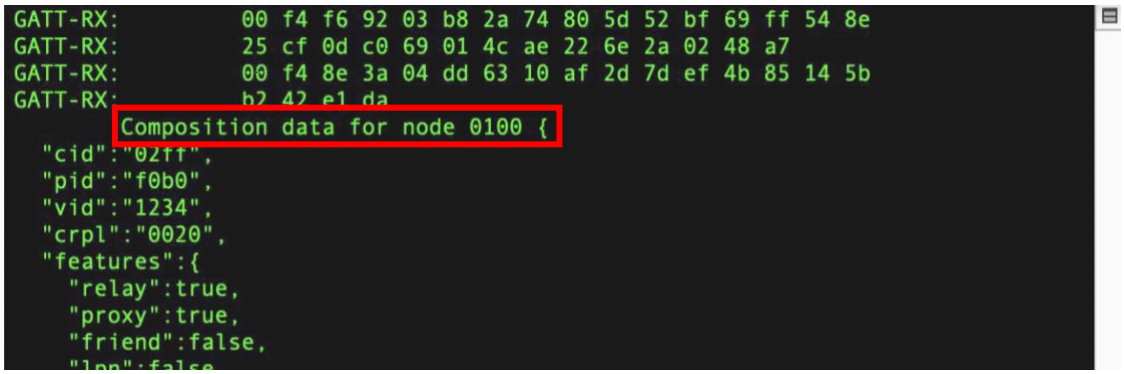
You also can stop the process of discovering unprovisioned devices by using the below command.

```
discover-unprovisioned off
```

Provision
Copy the “Device UUID” and paste it after the *provision* command, as shown below, to initiate the provisioning process.

```
provision <Device UUID>
```

If provisioning is successful, you can find the primary element address in the red box of “Composition data for node XXXX”, XXXX is the primary element.



Configuration
Since provisioning is completed, it’s time to perform model configuration. Type in below command to switch *menu config* submenu.

```
menu config
```

Select the target device to perform the model configuration.

```
target <primary_element_address>
```

Usually, model configuration operations include:

- add AppKey for the node
- bind element index, AppKey and target model
- publish/subscribe setup

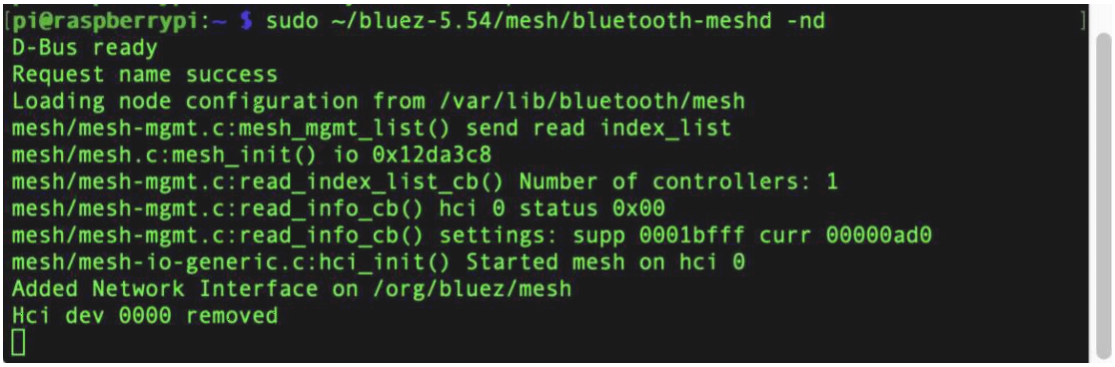
Please refer to the section of “meshctl available commands” to get the full commands list.

5.2 PB-ADV

mesh-cfgclient is a tool that provides Provisioner functionality and it’s over PB-ADV.

Launch mesh-cfgclient
On the RPI, we need to have full ownership of the controller, so stop *bluetoothd*: Stop *bluetoothd* and start *bluetooth-meshd.*

```
sudo systemctl stop bluetooth
sudo ~/bluez-5.54/mesh/bluetooth-meshd -nd
```



Open a new SSH and make it connect to Raspberry Pi board, type below commands and you will see *mesh-cfgclient* starts to work.

```
cd ~

mesh-cfgclient
```

When you run *mesh-cfgclient* for the first time, the tool will notify you of a warning:

"Warning: config file "/home/pi/.config/meshcfg/config_db.json" not found".

Use the below command to create a new mesh network[3].

```
create
```

Add AppKey and NetKey

```
appkey-create 0 0
```

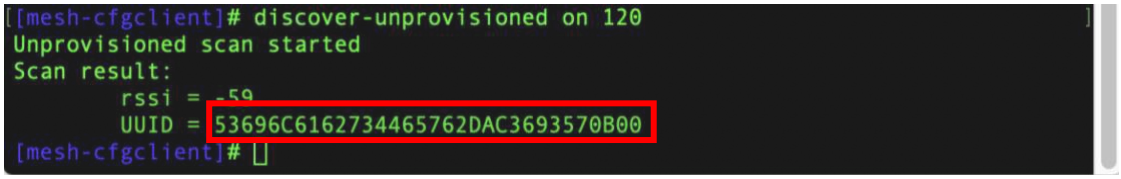
Discover the unprovisioned device

```
discover-unprovisioned <on/off> [seconds]
```

<on/off> mean start or stop discovering unprovisioned device.

[3] After that, no need to issue this *create* command again until you want to create a new mesh network.

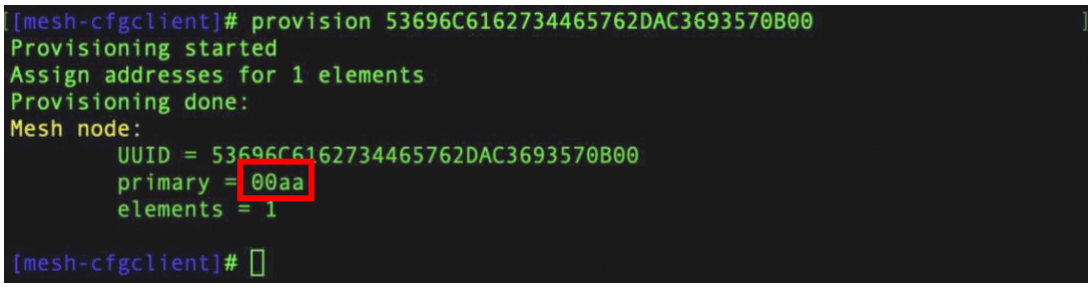
[seconds] means how many seconds the discovering process is going on.
When an unprovisioned device is found, the message will pop-up as below picture shown, the “UUID” in the red box will be used in the next section.



Provision
Copy the “UUID” and paste it after the *provision* command, as shown below, to initiate the provisioning process.

```
provision <UUID>
```

When provisioning is completed, you can find the primary element address in the red box of the below picture.



Configuration
Since provisioning is completed, it’s time to perform model configuration. Type in below command to switch *menu config* submenu.

```
menu config
```

Select the target device to perform the model configuration.

```
target <primary_element_address>
```

Usually, model configuration operations include:

- add AppKey for the node
- bind element index, AppKey, and target model
- publish/subscribe setup

Please refer to the section of “mesh-cfgclient available commands” to get the full commands list.

Appendix A - meshctl available commands

This section lists all the commands *meshctl* supports, *meshctl* has 3 command menus:

- main menu
- menu config
- menu onoff

menu main command list

Command	Description
config	Configuration Model Submenu
onoff	On/Off Model Submenu
list	List available controllers
show [ctrl]	Controller information
select <ctrl>	Select default controller
security [0(low)/1(medium)/2(high)]	Display or change provision security level
info [dev]	Device information
connect [net_idx] [dst]	Connect to mesh network or node on network
discover-unprovisioned <on/off>	Look for devices to provision
provision <uuid>	Initiate provisioning
power <on/off>	Set controller power
disconnect [dev]	Disconnect device
mesh-info	Mesh networkinfo (provisioner)
local-info	Local mesh node info
menu <name>	Select submenu
version	Display version
quit	Quit program
exit	Quit program
help	Display help about this program
export	Print environment variables

menu config command list

Command	Description
target <unicast>	Set target node to configure
composition-get [page_num]	Get composition data
netkey-add <net_idx>	Add network key
netkey-del <net_idx>	Delete network key
appkey-add <app_idx>	Add application key
appkey-del <app_idx>	Delete application key
bind <ele_idx> <app_idx> <mod_id> [cid]	Bind app key to a model
mod-appidx-get <ele_addr> <model id>	Get model app_idx
ttl-set <ttl>	Set default TTL
ttl-get	Get default TTL
pub-set <ele_addr> <pub_addr> <app_idx> <per (step|res)> <re-xmt (cnt|per)> <mod id> [cid]	Set publication
pub-get <ele_addr> <model>	Get publication
proxy-set <proxy>	Set proxy state
proxy-get	Get proxy state
ident-set <net_idx> <state>	Set node identity state
ident-get <net_idx>	Get node identity stat
beacon-set <state>	Set node identity state
beacon-get	Get node beacon state
relay-set <relay> <rexmt count> <rexmt steps>	Set relay
relay-get	Get relay
hb-pub-set <pub_addr> <count> <period> <ttl> <features> <net_idx>	Set heartbeat publish
hb-pub-get	Get heartbeat publish
hb-sub-set <src_addr> <dst_addr> <period>	Set heartbeat subscribe
hb-sub-get	Get heartbeat subscribe

sub-add <ele_addr> <sub_addr> <model id>	Add subscription
sub-get <ele_addr> <model id>	Get subscription
node-reset	Reset a node and remove it from network
back	Return to main menu
version	Display version
quit	Quit program
exit	Quit program
help	Display help about this program
export	Print environment variables

menu onoff command list

Command	Description
target <unicast>	Set node to configure
get	Get ON/OFF status
onoff <0/1>	Send "SET ON/OFF" command
back	Return to main menu
version	Display version
quit	Quit program
exit	Quit program
help	Display help about this program
export	Print environment variables

Appendix B - mesh-cfgclient command list

This section lists all the commands *mesh-cfgclient* supports, *mesh-cfgclient* has 2 command menus:

- main menu
- menu config

main menu command list

Command	Description
config	Configuration Model Submenu
create [unicast_range_low]	Create new mesh network with one initial node
discover-unprovisioned <on/off> [seconds]	Look for devices to provision
appkey-create <net_idx> <app_idx>	Create a new local AppKey
appkey-import <net_idx> <app_idx> <key>	Import a new local AppKey
appkey-update <app_idx>	Update local AppKey
appkey-delete <app_idx>	Delete local AppKey
subnet-create <net_idx>	Create a new local subnet (NetKey)
subnet-import <net_idx> <key>	Import a new local subnet (NetKey)
subnet-update <net_idx>	Update local subnet (NetKey)
subnet-delete <net_idx>	Delete local subnet (NetKey)
subnet-set-phase <net_idx> <phase>	Set subnet (NetKey) phase
list-unprovisioned	List unprovisioned devices
provision <uuid>	Initiate provisioning
node-import <uuid> <net_idx> <primary> <ele_count> <dev_key>	Import an externally provisioned remote node
node-delete <primary> <ele_count>	Delete a remote node
list-nodes	List remote mesh nodes
keys	List available keys
menu <name>	Select submenu
version	Display version
quit	Quit program
exit	Quit program
help	Display help about this program
export	Print environment variables

menu config command list

Command	Description
target <unicast>	Set target node to configure
timeout <seconds>	Set response timeout (seconds)
composition-get [page_num]	Get composition data
netkey-add <net_idx>	Add NetKey
netkey-update <net_idx>	Update NetKey
netkey-del <net_idx>	Delete NetKey
netkey-get	List NetKeys known to the node
appkey-add <app_idx>	Add AppKey
appkey-update <app_idx>	Add AppKey
appkey-del <app_idx>	Delete AppKey
appkey-get <net_idx>	List AppKeys bound to the NetKey
bind <ele_addr> <app_idx> <model_id> [vendor_id]	Bind AppKey to a model
unbind <ele_addr> <app_idx> <model_id> [vendor_id]	Remove AppKey from a model
mod-appidx-get <ele_addr> <model_id> [vendor_id]	Get model app_idx
ttl-set <ttl>	Set default TTL
ttl-get	Get default TTL
pub-set <ele_addr> <pub_addr> <app_idx> <per (step|res)> <re-xmt (cnt|per)> <model_id> [vendor_id]	Set publication
pub-get <ele_addr> <model_id> [vendor_id]	Get publication
proxy-set <proxy>	Set proxy state
proxy-get	Get proxy state
ident-set <net_idx> <state>	Set node identity state
ident-get <net_idx>	Get node identity state
beacon-set <state>	Set node identity state
beacon-get	Get node beacon state
relay-set <relay> <rexmt count> <rexmt steps>	Set relay

relay-get	Get relay
friend-set <state>	Set friend state
friend-get network-transmit-get	Get friend state Get network transmit state
network-transmit-set <count> <steps>	Set network transmit state
hb-pub-set <pub_addr> <count> <period> <ttl> <features> <net_idx>	Set heartbeat publish
hb-pub-get	Get heartbeat publish
hb-sub-set <src_addr> <dst_addr> <period>	Set heartbeat subscribe
hb-sub-get	Get heartbeat subscribe
virt-add	Generate and add a virtual label
group-list	Display existing group addresses and virtual labels
sub-add <ele_addr> <sub_addr> <model_id> [vendor]	Add subscription
sub-del <ele_addr> <sub_addr> <model_id> [vendor]	Delete subscription
sub-wrt <ele_addr> <sub_addr> <model_id> [vendor]	Overwrite subscription
sub-del-all <ele_addr> <model_id> [vendor]	Delete subscription
sub-get <ele_addr> <model_id> [vendor]	Get subscription
node-reset	Reset a node and remove it from network
back	Return to main menu
version	Display version
quit	Quit program
exit	Quit program
help	Display help about this program
export	Print environment variables