

FLSEMI DECT NR+ Python Samples: Tutorial

Version 1.0.0, 2026-10. FL SEMICONDUCTOR LLC (FLSEMI).

1. What these samples show

A DECT NR+ mesh carries data between radios. FLSEMI gateways put an app on top of it: a phone or computer connects to its gateway over Bluetooth LE and sends and receives through the mesh. Boards on a USB serial port (nRF9151 DK, nRF9131 EK running FLSEMI firmware) can take part directly.

```
app (phone/PC) <-- Bluetooth LE --> gateway <-- DECT NR+ mesh --> gateway <-- Bluetooth LE
--> app
program (PC)   <-- USB serial ----> board   <-- DECT NR+ mesh --> any node
```

These features run on top:

- **Text messages**, to one gateway or to everyone (the group).
- **Group push-to-talk**: half duplex, one talker at a time. The network's sink relays each frame once per hop to every member.
- **One-to-one call**: the two radios leave the mesh for a private network for the length of the call, then return.

All voice coding (Opus) and all encryption in the app happen in the app. The gateways and the radios carry opaque frames.

2. What you need

Gateways. At least two FLSEMI DECT NR+ gateways on one network, for the Bluetooth examples. One of them is the network's sink.

Boards (optional). One or more boards with FLSEMI DECT NR+ firmware, for the serial examples. A board shows up as two USB serial ports; the host interface is on the second one (1 Mbaud).

Computer. Tested on macOS (Apple silicon): every example with Python 3.14; the serial examples, codec and encryption also with Python 3.9. Windows and Linux use the same libraries (bleak, pyserial, libopus) but have not yet been tested by FLSEMI.

- macOS, Windows or Linux;
- Python 3.9 or later;
- Bluetooth LE.

Opus library.

- macOS: `brew install opus`
- Debian/Ubuntu: `sudo apt install libopus0`
- Windows: `opus.dll` on PATH

Python packages: `python3 -m pip install -r requirements.txt` (bleak, pyserial, cryptography).

Audio input. A speech WAV, 16 kHz mono 16-bit. Convert other files with: `ffmpeg -i in.mp3 -ac 1 -ar 16000 -sample_fmt s16 speech_16k.wav`

On macOS, run the Bluetooth examples from Terminal (or another app that has Bluetooth permission).

3. The group key

Encryption in the app uses one 16-byte key per group. Only the apps hold it; gateways, relays and the sink never do.

```
cd examples
python3 make_group_key.py group_key.json
```

Give the same file to every member's app, over a channel you trust. Every example takes it with `--key`. Without `--key`, messages and voice go in clear. The mesh's own link security still protects each hop, but relays and the sink can read the content.

A group can change key without breaking traffic. Each key carries a generation number (0–7), and an app keeps the previous generation for receiving while it sends with the new one.

4. Pairing a gateway

A gateway's Bluetooth name is `BFi53-` followed by its ID, for example `BFi53-1a2b3c4d`.

1. Press the gateway's **Button 1** once. The LED breathes blue: the pairing window is open.
2. Run any Bluetooth example with `--gw BFi53-...`. The operating system pairs (LE Secure Connections).
3. From then on the gateway reconnects to this computer by itself. A paired gateway advertises without its name, so the client remembers its address from the first link (`~/ .config/flsemi_samples/known_gateways.json`) and finds it by that address next time.

Keep in mind:

- **One connection at a time.** Each gateway serves one Bluetooth connection, so close other apps that hold it.
- **Holding Button 1 for 3 s clears all pairings.** If the computer then refuses to reconnect, remove the gateway from the operating system's Bluetooth settings and pair again.

5. The examples

Every example prints its usage with `--help`.

5.1 Text messages: ble_message.py

```
# on the computer linked to gateway B
python3 ble_message.py --gw BFi53-5e6f7a8b --key group_key.json listen 120
# on the computer linked to gateway A
python3 ble_message.py --gw BFi53-1a2b3c4d --key group_key.json send --to all "meet at the gate"
```

Expected at B: from 0x1a2b3c4d: meet at the gate.

Destinations. --to all reaches every gateway in the network. --to 0x... reaches one gateway.

Size. A message holds up to 200 octets. Encryption adds 16, which leaves 184 for text.

What the receiver prints:

- [in clear] for a message sent without a key;
- [encrypted, cannot open] for one sealed with a key it does not have.

5.2 Group push-to-talk: ble_group_voice.py

```
python3 ble_group_voice.py --talk BFi53-5e6f7a8b --listen BFi53-1a2b3c4d \
    --key group_key.json --wav speech_16k.wav --out heard.wav
```

One computer links to both gateways and plays both members:

1. Both join the group.
2. One talks the WAV in real time, Opus 12 kbit/s in 40 ms frames.
3. The other records what arrives into heard.wav. Missing frames are concealed.

The summary line gives frames talked, heard and concealed.

Packet size limit. A group frame must fit one occurrence of the sink's broadcast resource, so the example caps each Opus packet. Encrypted frames stay within 76 octets. Keep the cap in your own app.

5.3 One-to-one call: ble_call.py

```
python3 ble_call.py --caller BFi53-5e6f7a8b --callee BFi53-1a2b3c4d \
    --key group_key.json --wav speech_16k.wav --out heard_call.wav
```

Call set-up with --key:

1. The caller's app makes a session key and seals it with the group key.
2. Its gateway sends only the seal across the mesh.
3. The callee's app opens the seal and gives the key to its own gateway when it accepts.
4. The two radios then move to a private network.

Set-up takes a few seconds. Hanging up returns both radios to the mesh.

5.4 Broadcast between boards: `uart_broadcast.py`

```
python3 uart_broadcast.py --from /dev/cu.usbmodemAAAA3 --to /dev/cu.usbmodemBBBB3  
/dev/cu.usbmodemCCCC3
```

The first board broadcasts five datagrams; each listed board reports how many it received. On Windows the ports are `COM...`.

5.5 Messages from a board: `uart_message.py`

Same format as the gateway apps, so a board and a phone can message each other:

```
python3 uart_message.py --port /dev/cu.usbmodemBBBB3 --key group_key.json listen 60  
python3 uart_message.py --port /dev/cu.usbmodemAAAA3 --key group_key.json send --to all  
"board here"
```

5.6 A board in the group walkie-talkie: `uart_group_voice.py`

```
python3 uart_group_voice.py --port /dev/cu.usbmodemAAAA3 --key group_key.json listen 30  
python3 uart_group_voice.py --port /dev/cu.usbmodemAAAA3 --key group_key.json talk  
speech_16k.wav
```

The board announces itself to the sink every 20 s and then hears the group like any phone. While listening it records `heard_group_uart.wav`.

5.7 Voice between two boards: `uart_p2p_voice.py`

```
python3 uart_p2p_voice.py --talk /dev/cu.usbmodemAAAA3 --listen /dev/cu.usbmodemBBBB3 \  
--key group_key.json --wav speech_16k.wav
```

Voice frames go to the other board's node ID over the mesh.

Use point-to-point voice between neighbours only, one hop apart. A routed unicast stream of 25 frames per second does not keep up beyond one hop: on the bench, two hops delivered about a third of the frames. For voice across several hops, use the group walkie-talkie, which the sink relays once per hop, or a one-to-one call, which moves both radios onto a direct link.

6. Tested

These figures were measured on 2026-10-05 with this package on a bench network: Band 9, nine radios, five hops, and two FLSEMI gateways. A Mac stood in as the phone over Bluetooth LE. They are bench figures, not guarantees.

Example	Result
<code>ble_message.py</code>	B → A: sealed group text, sealed direct text and plain text, all three received and shown correctly
<code>ble_group_voice.py</code>	sealed, B → A, one hop: 301 of 301 frames, none concealed
<code>ble_call.py</code>	sealed invite: in call after 7.5 s, then 301 of 301 frames
<code>uart_broadcast.py</code>	from one board: 5/5 at nodes 3, 4 and 5 hops away
<code>uart_message.py</code>	a sealed group message opened at a board 5 hops away; a board's sealed message opened in a phone app (<code>ble_message.py</code>)
<code>uart_group_voice.py</code>	a board talking, another 4 hops below the sink listening: 275 of 302 frames heard
<code>uart_p2p_voice.py</code>	one hop: 300/301 and 297/301; two hops: 95/301 (see 5.7)

7. Building your own app

7.1 The gateway link (Bluetooth LE)

- **Service and characteristics:**
 - service `f1e5e000-7c1a-4b9e-9b1d-46a0b6f9d300`;
 - write requests to `...e001`, write without response;
 - subscribe to notifications on `...e002`.
- **Pairing.** Subscribing to the encrypted characteristic `da2e7828-fbce-4e01-ae9e-261174997c48` makes the operating system pair.
- **Frame format.** Both directions use `0xF7 | type | seq | length u16 LE | payload`. A request's answer carries the same `seq`; events from the mesh carry 0.

Type	Direction	Payload
0x01 hello	to gateway	— → 0x81 info: proto, flags (bit 0 sink), ID u32, firmware text
0x02 send	to gateway	destination u32 + data → 0x82 result (0 = queued)
0x03 who	to gateway	— → 0x83 who list
0x10 call start	to gateway	callee u32, carrier u16 (0 = auto) [, session key 16, seal 33]
0x11 call accept	to gateway	[session key 16]
0x12 reject / 0x13 hang up	to gateway	—
0x16 call voice	to gateway	voice frame (no answer unless refused)
0x18 group join	to gateway	1 = join, 0 = leave
0x19 group voice	to gateway	voice frame (no answer unless refused)
0x90 message	from gateway	source u32 + data
0x91 call state	from gateway	state, call ID u32, peer u32, info i16
0x93 call voice	from gateway	voice frame
0x94 group voice	from gateway	talker u32 + voice frame
0x95 floor	from gateway	holder u32, refused u8
0x96 call invite	from gateway	call ID u32, caller u32, seal (empty = not sealed)
0xFF error	from gateway	request type, error i16

Number formats. All multi-octet numbers on the link are little-endian, except where a payload says BE.

Destination IDs. 0xFFFFFFFF is everyone and 0 is the sink. A message from the sink shows the source 0xFFFFFFFFE.

7.2 Voice frames

```
flags u8 (bit 0 first, bit 1 last) | seq u16 LE | codec u8 (1 = Opus) | Opus packet
```

- Opus runs at 16 kHz mono, 12 kbit/s, 40 ms frames, VOIP mode.
- End a talk with an empty frame that has the last bit set.
- The sink reads only the flags octet, to run the floor.

7.3 Encryption in the app

The construction is AES-128-CTR, with a 5-octet CMAC MIC computed over the plaintext and ciphered with it. Every sub-key is CMAC(group key, label). The exact layouts and labels are in `flsemi_samples/app_crypto.py`.

Voice frame.

- The flags, seq and codec octets stay in clear, with bit 7 of the codec octet set and the key generation in bits 4–6.
- A 4-octet packet counter follows, then the ciphertext.
- The IV is talker ID, group ID and counter.

Message.

- 0xE2 | 1 | generation | sender ID | counter | ciphertext.
- The sender is in the header because the sink's broadcasts carry no source address. A receiver must check it against the source the network reports, whenever the network reports one.

Call invite seal.

- generation | nonce 8 | ciphered session key | tag 8.
- The tag binds the caller and the callee IDs.

Packet counter.

- It counts 10 ms clock ticks and never goes backwards. Keep it that way, so that a restarted app is not taken for a replay.
- Receivers keep a window of the last 1024 counters per sender.

7.4 Porting notes

Bluetooth LE.

- iOS/macOS: CoreBluetooth.
- Android: `BluetoothGatt`; request an MTU of 247 or more.

- Windows: WinRT `GattSession`.
- Write requests in chunks of MTU – 3 octets.

Opus.

- iOS/Android: link libopus.
- Web: WebCodecs or a WASM build of libopus.

AES-CTR and AES-CMAC.

- iOS: CommonCrypto for AES. CryptoKit has no CMAC, so build CMAC on AES-ECB per RFC 4493.
- Android: `javax.crypto` with the "AESCMAC" Mac on API 31+, or Bouncy Castle.

Audio timing. Send voice in real time, one frame per 40 ms, and play out through a small jitter buffer of about 3 frames.

8. Limits

- **One talker at a time** in a group. A second talker gets a `floor` event and is not relayed.
- **One Bluetooth connection per gateway.**
- **Message size:** 200 octets per message (184 of text when encrypted).
- **Group voice frame size:** group voice frames must stay within the packet cap (section 5.2).
- **Point-to-point voice between boards:** one hop only (section 5.7).
- **Range and node count:** this package states neither. Ask FLSEMI about your deployment.

9. Licence

Source code provided under the FLSEMI 5-Clause License (LICENSE). You may modify it, build it into commercial products and redistribute it, in products that incorporate an FLSEMI or BANFi DECT NR+ device. Keep the copyright notice in the source.

Copyright (c) 2026 FL SEMICONDUCTOR LLC (FLSEMI). Questions: flsemi.us.