# **BLE Resolvable Random Private Address (RPA)**

# **Example 1(Theory)**

### What is a Resolvable Private Address (RPA)?

A Resolvable Private Address (RPA) is a type of Bluetooth address used in BLE to enhance privacy. It changes periodically, making it difficult for unauthorized devices to track a BLE device over time. However, authorized devices (peers) that have a \*\*pre-shared key\*\* called the \*\*Identity Resolving Key (IRK)\*\* can resolve the RPA to determine the actual identity of the device.

---

### Key Components:

1. \*\*Resolvable Private Address (RPA)\*\*:

- A temporary, randomly generated address that changes frequently.

- Used to prevent tracking by unauthorized devices.

2. \*\*Identity Resolving Key (IRK)\*\*:

- A pre-shared secret key exchanged between two devices during pairing.

- Used to generate and resolve RPAs.

3. \*\*Public or Random Static Address\*\*:

- The actual address of the BLE device (either a public address or a random static address).

- This is the "real" address that the RPA ultimately resolves to.

---

### How It Works:

1. \*\*Generating the RPA\*\*:

- The BLE device generates a new RPA periodically using its IRK and a random number.

- The RPA is derived from a cryptographic hash function (e.g., AES) applied to the IRK and the random number.

2. \*\*Advertising with the RPA\*\*:

- The BLE device advertises using the RPA instead of its real address (public or random static address).

3. \*\*Resolving the RPA\*\*:

- When an authorized peer (e.g., a smartphone) receives the RPA, it uses its copy of the IRK to resolve the RPA back to the device’s real address.

- This allows the peer to recognize the device even though its address has changed.

---

### Example Scenario:

Imagine you have a \*\*BLE fitness tracker\*\* and a \*\*smartphone\*\* that are paired.

1. \*\*Pairing\*\*:

- During pairing, the fitness tracker and the smartphone exchange their \*\*IRKs\*\*.

- The fitness tracker’s real address is, say, `00:11:22:33:44:55` (a public address).

2. \*\*Generating the RPA\*\*:

- The fitness tracker generates a new RPA, e.g., `A1:B2:C3:D4:E5:F6`, using its IRK and a random number.

- This RPA changes every 15 minutes (or another interval) to prevent tracking.

3. \*\*Advertising\*\*:

- The fitness tracker starts advertising its presence using the RPA `A1:B2:C3:D4:E5:F6` instead of its real address.

4. \*\*Resolving the RPA\*\*:

- Your smartphone, which has the fitness tracker’s IRK, receives the RPA `A1:B2:C3:D4:E5:F6`.

- Using the IRK, the smartphone resolves the RPA back to the real address `00:11:22:33:44:55`.

- Now, the smartphone knows that the device advertising with `A1:B2:C3:D4:E5:F6` is actually your fitness tracker.

5. \*\*Privacy Benefit\*\*:

- If an unauthorized device (e.g., a hacker’s scanner) sees the RPA `A1:B2:C3:D4:E5:F6`, it cannot determine the real address of the fitness tracker because it doesn’t have the IRK.

- Even if the RPA changes to `B2:C3:D4:E5:F6:A1` later, the hacker cannot link it to the previous RPA or the real address.

---

### Why Use RPAs?

- \*\*Privacy\*\*: RPAs prevent unauthorized devices from tracking a BLE device over time.

- \*\*Security\*\*: Only devices with the correct IRK can resolve the RPA to the real address.

- \*\*Dynamic Addressing\*\*: RPAs change frequently, making it harder for attackers to correlate addresses.

---

### Summary:

- A \*\*Resolvable Private Address (RPA)\*\* is a temporary, changing address used by BLE devices to enhance privacy.

- The \*\*Identity Resolving Key (IRK)\*\* is used to generate and resolve RPAs.

- Authorized devices (with the IRK) can resolve the RPA to the device’s real address, while unauthorized devices cannot.

In the example, the fitness tracker uses an RPA to advertise, and your smartphone resolves it to the real address using the IRK. This ensures privacy and security in BLE communication.

# **Example 2 (Real Address Used)**

Sure! Let’s go through a \*\*step-by-step example\*\* with actual addresses to illustrate how a \*\*Resolvable Private Address (RPA)\*\* works in Bluetooth Low Energy (BLE). We’ll use realistic-looking addresses and walk through the process of generating and resolving an RPA.

---

### Scenario:

- \*\*BLE Device\*\*: A fitness tracker with a \*\*public address\*\* of `00:11:22:33:44:55`.

- \*\*Peer Device\*\*: A smartphone that has paired with the fitness tracker and exchanged \*\*Identity Resolving Keys (IRKs)\*\*.

---

### Step 1: Pairing and IRK Exchange

During the pairing process:

- The fitness tracker and smartphone exchange their \*\*IRKs\*\*.

- Let’s assume:

- Fitness tracker’s IRK: `A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8` (a 128-bit key).

- Smartphone’s IRK: `F8E7D6C5B4A3F2E1D0C9B8A7F6E5D4C3B2A1`.

---

### Step 2: Generating the RPA

The fitness tracker generates a new \*\*Resolvable Private Address (RPA)\*\* every 15 minutes (or another interval). Here’s how it works:

1. \*\*Random Number (prand)\*\*:

- The fitness tracker generates a 24-bit random number called `prand`. Let’s say:

- `prand = 0xA3B4C5`.

2. \*\*Hash Calculation\*\*:

- The fitness tracker uses its IRK and the `prand` to compute a 24-bit hash using a cryptographic function (e.g., AES).

- The formula is: `hash = AES(IRK, prand)` (only the first 24 bits of the result are used).

- Let’s assume the computed hash is `0x1E2F3D`.

3. \*\*Constructing the RPA\*\*:

- The RPA is constructed by combining the `prand` and the `hash`:

- The first 24 bits are the `prand` (`0xA3B4C5`).

- The last 24 bits are the `hash` (`0x1E2F3D`).

- The resulting RPA is: `A3:B4:C5:1E:2F:3D`.

---

### Step 3: Advertising with the RPA

The fitness tracker starts advertising using the RPA `A3:B4:C5:1E:2F:3D` instead of its real public address (`00:11:22:33:44:55`).

---

### Step 4: Resolving the RPA

When the smartphone receives the RPA `A3:B4:C5:1E:2F:3D`, it uses its copy of the fitness tracker’s IRK to resolve the RPA back to the real address.

1. \*\*Extract `prand` and `hash`\*\*:

- The smartphone extracts the first 24 bits of the RPA as `prand`:

- `prand = 0xA3B4C5`.

- The last 24 bits are the `hash`:

- `hash = 0x1E2F3D`.

2. \*\*Recompute the Hash\*\*:

- The smartphone uses the fitness tracker’s IRK (`A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8`) and the `prand` (`0xA3B4C5`) to recompute the hash:

- `computed\_hash = AES(IRK, prand)`.

- Let’s assume the computed hash matches the received hash: `0x1E2F3D`.

3. \*\*Verify the RPA\*\*:

- If the computed hash matches the received hash, the smartphone confirms that the RPA belongs to the fitness tracker.

4. \*\*Resolve to the Real Address\*\*:

- The smartphone now knows that the RPA `A3:B4:C5:1E:2F:3D` corresponds to the fitness tracker’s real public address: `00:11:22:33:44:55`.

---

### Step 5: Privacy in Action

- If an unauthorized device (e.g., a hacker’s scanner) sees the RPA `A3:B4:C5:1E:2F:3D`, it cannot determine the real address of the fitness tracker because it doesn’t have the IRK.

- Even if the RPA changes to a new value (e.g., `B2:C3:D4:E5:F6:A1`) after 15 minutes, the hacker cannot link it to the previous RPA or the real address.

---

### Summary of Addresses:

- \*\*Real Public Address\*\*: `00:11:22:33:44:55` (fitness tracker’s actual address).

- \*\*RPA\*\*: `A3:B4:C5:1E:2F:3D` (temporary address generated using the IRK and `prand`).

- \*\*IRK\*\*: `A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8` (used to generate and resolve the RPA).

---

### Key Takeaways:

- The RPA is a temporary, changing address that enhances privacy.

- Only devices with the correct IRK can resolve the RPA to the real address.

- Unauthorized devices cannot track or identify the BLE device because they lack the IRK.

This example demonstrates how RPAs and IRKs work together to protect the privacy of BLE devices.

|-------------------------------------------------END----------------------------------------------------------------------------|

# **Example 1 (BLE Adv Packet)**

Sure! Let’s break down a \*\*BLE Advertisement Packet\*\* with an example, including the structure, addresses, and data. BLE advertising packets are used by devices to broadcast their presence and provide information to nearby devices.

---

### BLE Advertisement Packet Structure

A BLE advertisement packet consists of the following fields:

1. \*\*Preamble\*\* (1 byte):

- Used for synchronization. Always `0xAA` for advertising channels.

2. \*\*Access Address\*\* (4 bytes):

- Fixed value for advertising packets: `0x8E89BED6`.

3. \*\*PDU (Protocol Data Unit)\*\*:

- Contains the actual advertising data.

- Includes:

- \*\*Header\*\* (2 bytes):

- Contains information like the PDU type, TxAdd/RxAdd flags, and length.

- \*\*Payload\*\* (up to 31 bytes):

- Contains the actual advertising data, such as the device address, flags, and service UUIDs.

4. \*\*CRC\*\* (3 bytes):

- Cyclic Redundancy Check for error detection.

---

### Example BLE Advertisement Packet

Let’s create an example of a BLE advertisement packet for a fitness tracker with the following details:

- \*\*Device Address\*\*: `00:11:22:33:44:55` (public address).

- \*\*Flags\*\*: `0x06` (LE General Discoverable Mode, BR/EDR Not Supported).

- \*\*Service UUID\*\*: `0x180D` (Heart Rate Service).

- \*\*Device Name\*\*: `FitnessTracker`.

---

### Step-by-Step Breakdown of the Packet

#### 1. Preamble:

- Always `0xAA` for advertising packets.

#### 2. Access Address:

- Fixed value: `0x8E89BED6`.

#### 3. PDU (Protocol Data Unit):

- \*\*Header\*\* (2 bytes):

- PDU Type: `0x00` (ADV\_IND, connectable and scannable).

- TxAdd: `0x00` (public address).

- RxAdd: `0x00` (not used in advertising).

- Length: `0x1A` (26 bytes of payload).

- \*\*Payload\*\* (26 bytes):

- \*\*Flags\*\* (3 bytes):

- Length: `0x02` (2 bytes of data).

- Type: `0x01` (Flags).

- Value: `0x06` (LE General Discoverable Mode, BR/EDR Not Supported).

- \*\*Service UUID\*\* (5 bytes):

- Length: `0x03` (3 bytes of data).

- Type: `0x03` (Complete List of 16-bit Service UUIDs).

- Value: `0x180D` (Heart Rate Service).

- \*\*Device Name\*\* (18 bytes):

- Length: `0x10` (16 bytes of data).

- Type: `0x09` (Complete Local Name).

- Value: `FitnessTracker` (16 characters, padded if necessary).

#### 4. CRC:

- Calculated based on the PDU. Let’s assume it’s `0x123456` for this example.

---

### Final Advertisement Packet

Here’s the complete BLE advertisement packet in hexadecimal format:

```

AA 8E 89 BE D6 00 1A 02 01 06 03 03 0D 18 10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72 12 34 56

```

---

### Explanation of the Packet

1. \*\*Preamble\*\*:

- `AA` (synchronization).

2. \*\*Access Address\*\*:

- `8E 89 BE D6` (fixed value for advertising).

3. \*\*PDU\*\*:

- \*\*Header\*\*:

- `00` (PDU Type: ADV\_IND).

- `1A` (Length: 26 bytes of payload).

- \*\*Payload\*\*:

- Flags:

- `02 01 06` (Length: 2, Type: Flags, Value: LE General Discoverable Mode).

- Service UUID:

- `03 03 0D 18` (Length: 3, Type: Complete List of 16-bit Service UUIDs, Value: Heart Rate Service).

- Device Name:

- `10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72` (Length: 16, Type: Complete Local Name, Value: "FitnessTracker").

4. \*\*CRC\*\*:

- `12 34 56` (example CRC value).

---

### Summary of the Packet

- \*\*Preamble\*\*: `AA`

- \*\*Access Address\*\*: `8E 89 BE D6`

- \*\*PDU Header\*\*: `00 1A`

- \*\*Payload\*\*:

- Flags: `02 01 06`

- Service UUID: `03 03 0D 18`

- Device Name: `10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72`

- \*\*CRC\*\*: `12 34 56`

---

### Key Points:

- The packet advertises the device as \*\*connectable\*\* and \*\*scannable\*\*.

- It includes the \*\*flags\*\*, \*\*service UUID\*\*, and \*\*device name\*\*.

- The payload is limited to 31 bytes, so the data must be concise.

This example demonstrates how a BLE advertisement packet is structured and how it conveys information about the device.

# **Example 2 (BLE Data Packet)**

Sure! Let’s dive into \*\*BLE Data Packets\*\* with an example. BLE data packets are used for communication between two connected devices (e.g., a fitness tracker and a smartphone). These packets carry actual data, such as sensor readings, commands, or notifications.

---

### BLE Data Packet Structure

A BLE data packet consists of the following fields:

1. \*\*Preamble\*\* (1 byte):

- Used for synchronization. Always `0xAA` for data channels.

2. \*\*Access Address\*\* (4 bytes):

- Unique address for the connection between two devices. It is generated during the connection setup.

3. \*\*PDU (Protocol Data Unit)\*\*:

- Contains the actual data.

- Includes:

- \*\*Header\*\* (2 bytes):

- Contains information like the LLID (Logical Link Identifier), NESN (Next Expected Sequence Number), SN (Sequence Number), and length.

- \*\*Payload\*\* (up to 27 bytes for BLE 4.x/5.0):

- Contains the actual data being transmitted.

4. \*\*CRC\*\* (3 bytes):

- Cyclic Redundancy Check for error detection.

---

### Example Scenario

Let’s assume:

- \*\*Fitness Tracker\*\* (Peripheral) is connected to a \*\*Smartphone\*\* (Central).

- The fitness tracker is sending a \*\*heart rate measurement\*\* to the smartphone.

- The heart rate value is `75 bpm`.

---

### Step-by-Step Breakdown of the Data Packet

#### 1. Preamble:

- Always `0xAA` for data packets.

#### 2. Access Address:

- Unique to the connection. Let’s assume it’s `0x12345678`.

#### 3. PDU (Protocol Data Unit):

- \*\*Header\*\* (2 bytes):

- LLID: `0x02` (Data PDU, L2CAP continuation or start of an L2CAP message).

- NESN: `0x1` (Next Expected Sequence Number).

- SN: `0x0` (Sequence Number).

- Length: `0x05` (5 bytes of payload).

- \*\*Payload\*\* (5 bytes):

- \*\*Attribute Opcode\*\* (1 byte):

- `0x1B` (Notification, indicating that the server is sending data to the client).

- \*\*Attribute Handle\*\* (2 bytes):

- `0x0012` (Handle for the Heart Rate Measurement characteristic).

- \*\*Heart Rate Value\*\* (2 bytes):

- `0x004B` (75 in hexadecimal).

#### 4. CRC:

- Calculated based on the PDU. Let’s assume it’s `0xABCDEF` for this example.

---

### Final Data Packet

Here’s the complete BLE data packet in hexadecimal format:

```

AA 12 34 56 78 02 05 1B 00 12 00 4B AB CD EF

```

---

### Explanation of the Packet

1. \*\*Preamble\*\*:

- `AA` (synchronization).

2. \*\*Access Address\*\*:

- `12 34 56 78` (unique to the connection).

3. \*\*PDU\*\*:

- \*\*Header\*\*:

- `02` (LLID: Data PDU, L2CAP continuation or start of an L2CAP message).

- `05` (Length: 5 bytes of payload).

- \*\*Payload\*\*:

- Attribute Opcode: `1B` (Notification).

- Attribute Handle: `00 12` (Handle for the Heart Rate Measurement characteristic).

- Heart Rate Value: `00 4B` (75 in hexadecimal).

4. \*\*CRC\*\*:

- `AB CD EF` (example CRC value).

---

### Summary of the Packet

- \*\*Preamble\*\*: `AA`

- \*\*Access Address\*\*: `12 34 56 78`

- \*\*PDU Header\*\*: `02 05`

- \*\*Payload\*\*:

- Attribute Opcode: `1B`

- Attribute Handle: `00 12`

- Heart Rate Value: `00 4B`

- \*\*CRC\*\*: `AB CD EF`

---

### Key Points:

- The packet is sent from the fitness tracker (peripheral) to the smartphone (central).

- It uses a \*\*notification\*\* to send the heart rate measurement.

- The payload contains the \*\*attribute handle\*\* and the \*\*heart rate value\*\*.

- The packet is protected by a \*\*CRC\*\* for error detection.

---

### Another Example: Write Request

Let’s say the smartphone wants to write a value to the fitness tracker (e.g., set a user profile).

#### Packet Details:

- \*\*Preamble\*\*: `AA`

- \*\*Access Address\*\*: `12 34 56 78`

- \*\*PDU Header\*\*: `02 07` (LLID: Data PDU, Length: 7 bytes).

- \*\*Payload\*\*:

- Attribute Opcode: `0x12` (Write Request).

- Attribute Handle: `0x0015` (Handle for the User Profile characteristic).

- Value: `0x01` (Enable profile).

- \*\*CRC\*\*: `0x987654` (example).

#### Final Packet:

```

AA 12 34 56 78 02 07 12 00 15 01 98 76 54

```

---

### Summary of Data Packets:

- Data packets are used for communication between connected devices.

- They include a \*\*header\*\*, \*\*payload\*\*, and \*\*CRC\*\*.

- The payload contains the actual data, such as sensor readings, commands, or notifications.

These examples demonstrate how BLE data packets are structured and used for communication in a connected BLE system.

|-------------------------------------------------END----------------------------------------------------------------------------|

# **BLE Packet for Advertisement**

Sure! Let’s break down the \*\*header of a BLE Advertisement Packet\*\* in detail. The header is part of the \*\*PDU (Protocol Data Unit)\*\* and contains important information about the advertisement packet, such as its type, addressing, and length.

---

### BLE Advertisement Packet Header Structure

The header of a BLE advertisement packet is \*\*2 bytes (16 bits)\*\* long and consists of the following fields:

1. \*\*PDU Type (4 bits)\*\*:

- Specifies the type of advertisement packet.

- Common types include:

- `0000`: ADV\_IND (Connectable and Scannable Undirected Advertising).

- `0001`: ADV\_DIRECT\_IND (Connectable Directed Advertising).

- `0010`: ADV\_NONCONN\_IND (Non-Connectable and Non-Scannable Advertising).

- `0110`: ADV\_SCAN\_IND (Scannable Undirected Advertising).

2. \*\*RFU (Reserved for Future Use) (1 bit)\*\*:

- Reserved for future use. Set to `0`.

3. \*\*ChSel (1 bit)\*\*:

- Indicates whether the device supports \*\*LE Channel Selection Algorithm #2\*\*.

- `0`: Not supported.

- `1`: Supported.

4. \*\*TxAdd (1 bit)\*\*:

- Indicates whether the \*\*transmitter’s address\*\* is public or random.

- `0`: Public address.

- `1`: Random address.

5. \*\*RxAdd (1 bit)\*\*:

- Indicates whether the \*\*receiver’s address\*\* is public or random (used in directed advertising).

- `0`: Public address.

- `1`: Random address.

6. \*\*Length (6 bits)\*\*:

- Specifies the length of the \*\*payload\*\* in bytes (up to 37 bytes for BLE 5.0, or 31 bytes for BLE 4.x).

---

### Example: BLE Advertisement Packet Header

Let’s create an example header for a \*\*connectable and scannable undirected advertisement (ADV\_IND)\*\*:

- \*\*PDU Type\*\*: `0000` (ADV\_IND).

- \*\*RFU\*\*: `0`.

- \*\*ChSel\*\*: `1` (LE Channel Selection Algorithm #2 supported).

- \*\*TxAdd\*\*: `0` (transmitter is using a public address).

- \*\*RxAdd\*\*: `0` (not applicable for undirected advertising, set to `0`).

- \*\*Length\*\*: `0x1A` (26 bytes of payload).

#### Binary Representation:

```

PDU Type: 0000

RFU: 0

ChSel: 1

TxAdd: 0

RxAdd: 0

Length: 011010 (26 in decimal)

```

Combining these fields into a 2-byte header:

```

0000 0 1 0 0 011010

```

#### Hexadecimal Representation:

- The binary `0000 0100 011010` translates to `0x041A` in hexadecimal.

---

### Full Advertisement Packet Example

Let’s assume the following details for the advertisement packet:

- \*\*Device Address\*\*: `00:11:22:33:44:55` (public address).

- \*\*Flags\*\*: `0x06` (LE General Discoverable Mode, BR/EDR Not Supported).

- \*\*Service UUID\*\*: `0x180D` (Heart Rate Service).

- \*\*Device Name\*\*: `FitnessTracker`.

#### Packet Breakdown:

1. \*\*Preamble\*\*: `0xAA` (fixed for BLE).

2. \*\*Access Address\*\*: `0x8E89BED6` (fixed for advertising packets).

3. \*\*PDU Header\*\*: `0x041A` (as calculated above).

4. \*\*Payload\*\*:

- Flags: `0x02 0x01 0x06`.

- Service UUID: `0x03 0x03 0x0D 0x18`.

- Device Name: `0x10 0x09 0x46 0x69 0x74 0x6E 0x65 0x73 0x73 0x54 0x72 0x61 0x63 0x6B 0x65 0x72`.

5. \*\*CRC\*\*: `0x123456` (example value).

#### Final Packet in Hexadecimal:

```

AA 8E 89 BE D6 04 1A 02 01 06 03 03 0D 18 10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72 12 34 56

```

---

### Explanation of the Header Fields in the Example

1. \*\*PDU Type\*\*: `0000` (ADV\_IND, connectable and scannable undirected advertising).

2. \*\*RFU\*\*: `0` (reserved, set to `0`).

3. \*\*ChSel\*\*: `1` (LE Channel Selection Algorithm #2 supported).

4. \*\*TxAdd\*\*: `0` (transmitter is using a public address).

5. \*\*RxAdd\*\*: `0` (not applicable for undirected advertising).

6. \*\*Length\*\*: `0x1A` (26 bytes of payload).

---

### Key Takeaways:

- The \*\*header\*\* of a BLE advertisement packet is \*\*2 bytes\*\* long.

- It contains critical information about the packet type, addressing, and payload length.

- The \*\*PDU Type\*\* determines the purpose of the advertisement (e.g., connectable, scannable, etc.).

- The \*\*TxAdd\*\* and \*\*RxAdd\*\* fields indicate whether the transmitter and receiver are using public or random addresses.

- The \*\*Length\*\* field specifies the size of the payload (up to 37 bytes for BLE 5.0 or 31 bytes for BLE 4.x).

This example demonstrates how the header is constructed and how it fits into a complete BLE advertisement packet.

|-------------------------------------------------END----------------------------------------------------------------------------|

# **Payload of BLE Advertisement Packet**

Certainly! Let’s break down the \*\*payload of a BLE Advertising Packet\*\* in detail, focusing on how it is divided into two sections: the \*\*Advertiser’s Address (AdvA)\*\* and the \*\*Advertisement Data (AdvData)\*\*. We’ll also provide an example to illustrate this structure.

---

### \*\*BLE Advertising Packet Payload Structure\*\*

The payload of a BLE advertising packet is divided into two main sections:

1. \*\*Advertiser’s Address (AdvA)\*\*:

- \*\*Length\*\*: 6 bytes.

- Contains the Bluetooth address of the device sending the advertisement.

- The address can be either a \*\*public address\*\* (assigned by the manufacturer) or a \*\*random address\*\* (temporary or private).

2. \*\*Advertisement Data (AdvData)\*\*:

- \*\*Length\*\*: Up to 31 bytes (for BLE 4.x) or 37 bytes (for BLE 5.0).

- Contains structured data in the form of \*\*AD (Advertising Data) structures\*\*.

- Each AD structure consists of:

- \*\*Length\*\* (1 byte): Length of the data field (excluding the length byte itself).

- \*\*Type\*\* (1 byte): Indicates the type of data (e.g., flags, service UUIDs, device name).

- \*\*Data\*\* (variable length): The actual data.

---

### \*\*Example: BLE Advertising Packet Payload\*\*

Let’s assume:

- The \*\*advertiser’s address\*\* is a \*\*public address\*\*: `00:11:22:33:44:55`.

- The \*\*advertisement data\*\* includes:

- \*\*Flags\*\*: `0x06` (LE General Discoverable Mode, BR/EDR Not Supported).

- \*\*Service UUID\*\*: `0x180D` (Heart Rate Service).

- \*\*Device Name\*\*: `FitnessTracker`.

#### Step-by-Step Breakdown of the Payload

1. \*\*Advertiser’s Address (AdvA)\*\*:

- The address is `00:11:22:33:44:55`.

- In the payload, it is represented as: `00 11 22 33 44 55`.

2. \*\*Advertisement Data (AdvData)\*\*:

- The advertisement data is structured as follows:

- \*\*Flags\*\*:

- Length: `0x02` (2 bytes of data).

- Type: `0x01` (Flags).

- Value: `0x06` (LE General Discoverable Mode, BR/EDR Not Supported).

- AD Structure: `02 01 06`.

- \*\*Service UUID\*\*:

- Length: `0x03` (3 bytes of data).

- Type: `0x03` (Complete List of 16-bit Service UUIDs).

- Value: `0x180D` (Heart Rate Service).

- AD Structure: `03 03 0D 18`.

- \*\*Device Name\*\*:

- Length: `0x10` (16 bytes of data).

- Type: `0x09` (Complete Local Name).

- Value: `FitnessTracker` (16 characters, padded if necessary).

- AD Structure: `10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72`.

- Combining these AD structures, the AdvData section is:

```

02 01 06 03 03 0D 18 10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72

```

3. \*\*Complete Payload\*\*:

- Combining the \*\*AdvA\*\* and \*\*AdvData\*\* sections, the complete payload is:

```

00 11 22 33 44 55 02 01 06 03 03 0D 18 10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72

```

---

### \*\*Final Advertising Packet\*\*

Let’s assume the following for the full advertising packet:

- \*\*Preamble\*\*: `0xAA` (fixed for BLE).

- \*\*Access Address\*\*: `0x8E89BED6` (fixed for advertising packets).

- \*\*PDU Header\*\*: `0x001A` (ADV\_IND, length: 26 bytes of payload).

- \*\*CRC\*\*: `0x123456` (example value).

#### Final Packet in Hexadecimal:

```

AA 8E 89 BE D6 00 1A 00 11 22 33 44 55 02 01 06 03 03 0D 18 10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72 12 34 56

```

---

### \*\*Explanation of the Payload\*\*

1. \*\*Advertiser’s Address (AdvA)\*\*:

- `00 11 22 33 44 55` (public address of the advertiser).

2. \*\*Advertisement Data (AdvData)\*\*:

- \*\*Flags\*\*:

- `02 01 06` (LE General Discoverable Mode, BR/EDR Not Supported).

- \*\*Service UUID\*\*:

- `03 03 0D 18` (Heart Rate Service).

- \*\*Device Name\*\*:

- `10 09 46 69 74 6E 65 73 73 54 72 61 63 6B 65 72` ("FitnessTracker").

---

### \*\*Key Takeaways\*\*

- The \*\*payload\*\* of a BLE advertising packet is divided into two sections:

- \*\*Advertiser’s Address (AdvA)\*\*: 6 bytes, containing the Bluetooth address of the advertiser.

- \*\*Advertisement Data (AdvData)\*\*: Up to 31 bytes (BLE 4.x) or 37 bytes (BLE 5.0), containing structured data in AD structures.

- Each \*\*AD structure\*\* consists of a \*\*length\*\*, \*\*type\*\*, and \*\*data\*\* field.

- The \*\*AdvData\*\* section can include information like flags, service UUIDs, and the device name.

This example demonstrates how the payload of a BLE advertising packet is structured and how it conveys information about the advertiser.

# **Bluetooth BR/EDR**

\*\*Bluetooth (BR/EDR)\*\* refers to \*\*Bluetooth Basic Rate/Enhanced Data Rate\*\*, which is the classic version of Bluetooth technology. It is the original Bluetooth standard and is widely used for applications like audio streaming (e.g., wireless headphones), file transfer, and peripheral device connectivity (e.g., keyboards, mice).

Let’s break down what \*\*BR/EDR\*\* means and how it works:

---

### \*\*What is Bluetooth (BR/EDR)?\*\*

1. \*\*BR (Basic Rate)\*\*:

- The original Bluetooth standard introduced in \*\*Bluetooth 1.0\*\*.

- Provides a data rate of \*\*1 Mbps\*\*.

- Uses \*\*Gaussian Frequency Shift Keying (GFSK)\*\* modulation.

- Designed for low-power, short-range communication.

2. \*\*EDR (Enhanced Data Rate)\*\*:

- Introduced in \*\*Bluetooth 2.0\*\* to improve data throughput.

- Provides higher data rates of \*\*2 Mbps\*\* (EDR2) and \*\*3 Mbps\*\* (EDR3).

- Uses more advanced modulation schemes like \*\*π/4-DQPSK\*\* and \*\*8DPSK\*\*.

- Maintains backward compatibility with BR.

---

### \*\*Key Features of Bluetooth (BR/EDR)\*\*

1. \*\*Data Rates\*\*:

- BR: 1 Mbps.

- EDR: 2 Mbps or 3 Mbps.

2. \*\*Modulation Schemes\*\*:

- BR: GFSK.

- EDR: π/4-DQPSK (for 2 Mbps) and 8DPSK (for 3 Mbps).

3. \*\*Frequency Band\*\*:

- Operates in the \*\*2.4 GHz ISM band\*\* (2400–2483.5 MHz).

4. \*\*Channels\*\*:

- Uses \*\*79 channels\*\*, each spaced 1 MHz apart.

- Employs \*\*frequency hopping\*\* (1600 hops per second) to reduce interference.

5. \*\*Power Consumption\*\*:

- Higher power consumption compared to \*\*Bluetooth Low Energy (BLE)\*\*.

- Suitable for applications requiring continuous data transfer (e.g., audio streaming).

6. \*\*Use Cases\*\*:

- Audio streaming (e.g., wireless headphones, speakers).

- File transfer.

- Peripheral device connectivity (e.g., keyboards, mice, printers).

---

### \*\*How Bluetooth (BR/EDR) Works\*\*

1. \*\*Connection Setup\*\*:

- Devices discover each other using \*\*inquiry\*\* and \*\*paging\*\* procedures.

- Once connected, they form a \*\*piconet\*\* (one master and up to seven active slaves).

2. \*\*Frequency Hopping\*\*:

- Devices hop between 79 channels in a pseudo-random sequence to avoid interference.

3. \*\*Data Transfer\*\*:

- Data is transmitted in packets over the selected channel.

- BR uses GFSK modulation, while EDR uses π/4-DQPSK or 8DPSK for higher data rates.

4. \*\*Power Management\*\*:

- Devices can enter low-power modes like \*\*sniff\*\*, \*\*hold\*\*, and \*\*park\*\* to save energy.

---

### \*\*Bluetooth (BR/EDR) vs. Bluetooth Low Energy (BLE)\*\*

| Feature | Bluetooth (BR/EDR) | Bluetooth Low Energy (BLE) |

|------------------------|-----------------------------|----------------------------|

| \*\*Data Rate\*\* | 1–3 Mbps | 1 Mbps |

| \*\*Power Consumption\*\* | Higher | Lower |

| \*\*Modulation\*\* | GFSK, π/4-DQPSK, 8DPSK | GFSK |

| \*\*Use Cases\*\* | Audio, file transfer | IoT, sensors, beacons |

| \*\*Frequency Hopping\*\* | 79 channels, 1600 hops/s | 40 channels, slower hopping|

---

### \*\*Example of Bluetooth (BR/EDR) in Action\*\*

Imagine you’re using a \*\*wireless headset\*\* to listen to music from your smartphone:

1. \*\*Pairing\*\*:

- The headset and smartphone are paired using Bluetooth (BR/EDR).

2. \*\*Connection\*\*:

- They form a piconet, with the smartphone as the master and the headset as the slave.

3. \*\*Audio Streaming\*\*:

- The smartphone transmits audio data to the headset at 2 Mbps (EDR).

4. \*\*Frequency Hopping\*\*:

- The devices hop between channels to avoid interference from Wi-Fi or other Bluetooth devices.

---

### \*\*Key Takeaways\*\*

- \*\*Bluetooth (BR/EDR)\*\* is the classic Bluetooth standard, offering data rates of 1–3 Mbps.

- It is optimized for applications like audio streaming and file transfer.

- BR/EDR uses frequency hopping and advanced modulation schemes to improve performance.

- While it consumes more power than BLE, it is ideal for continuous data transfer use cases.

This technology is the foundation of many everyday wireless devices and continues to be widely used alongside \*\*Bluetooth Low Energy (BLE)\*\*.