USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

BBC: Enabling BLE to Support Bluetooth Classic

Hsun-Wei Cho and Kang G. Shin, *University of Michigan*

https://www.usenix.org/conference/nsdi26/presentation/cho

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

BBC: Enabling BLE to Support Bluetooth Classic

Hsun-Wei Cho
University of Michigan

Kang G. Shin
University of Michigan

Abstract

Bluetooth Classic has been the technology used by the overwhelming majority of wireless headphones. However, Bluetooth Classic is incompatible with Bluetooth Low Energy (BLE), and hence cannot directly communicate with BLE devices. With the recent shift toward BLE, this incompatibility prevents using simple, energy-efficient BLE chips with Bluetooth headphones, and requires using more complex dual-mode chips to support both Bluetooth Classic and BLE.

To overcome this incompatibility, we present `BBC`, which enables Bluetooth-Classic connectivity on BLE chips. `BBC` sends and receives raw FSK bits using BLE hardware while emulating all other Bluetooth-Classic operations in the driver. By eliminating the need for Bluetooth-Classic hardware, `BBC` enables future devices to use BLE-only chips while maintaining the Bluetooth-Classic compatibility via emulation. It also enables new connectivity for current BLE devices to directly stream audio to Bluetooth-Classic headphones. `BBC` achieves a throughput of 557kbps and a packet error rate (PER) of 4.86% at the distance of 10m,[1] and provides the same audio quality as off-the-shelf Bluetooth-Classic chips.

1 Introduction

Since its invention more than 20 years ago, Bluetooth has been widely used for short-range, low-power communications. One major Bluetooth application is streaming audio to wireless headphones. Each year, more than one billion (more precisely, 1.01 billion [2] in 2024) Bluetooth audio streaming devices have been shipped worldwide and Bluetooth has been the dominant wireless technology for audio streaming. To stream high-fidelity audio, the overwhelming majority ($>$ 85% [3]) of headphones use the "Bluetooth-Classic" protocol.

The Bluetooth SIG (Special Interest Group) has been transitioning from Bluetooth Classic to Bluetooth Low Energy (BLE), since BLE enables simpler hardware implementation and higher energy-efficiency, and Bluetooth Classic is no longer in active development. However, BLE is an entirely separate and different protocol from Bluetooth Classic. BLE uses different bit-processing, frequency-hopping, timing, packet format from Bluetooth Classic and does not support A2DP (Advanced Audio Distribution Profile [4]) at all. Consequently, BLE chips cannot directly communicate with Bluetooth-Classic devices. Even though BLE single-mode chips are widely available (e.g., in ultra-low power BLE tags and BLE locks) in the market and are smaller, cheaper and highly energy-efficient (supporting years of battery life), they cannot be used with Bluetooth-Classic headphones. On the other hand, because the vast majority of headphones use Bluetooth Classic, multimedia devices must use *dual-mode* chips that contain both Bluetooth Classic and BLE hardware components. As shown in Bluetooth SIG's block diagram [5, 6], Bluetooth Classic and BLE are very different in each layer and share very little in common. As a result, dual-mode chips have to maintain two full and separate communication stacks (from digital circuits to application profiles) [7], thus increasing their cost, size and power consumption.

Figure 1: `BBC` enables direct communication between BLE chips and Bluetooth-Classic headphones

This incompatibility between BLE and Bluetooth Classic is becoming a major obstacle as the number of BLE devices continues to grow rapidly. As the industry shifts towards more and more BLE devices, billions of Bluetooth-Classic headphones would not be able to communicate with energy-efficient, ultra-low-power BLE devices.

[1] The maximum operating range of standard Class 2 devices (4dBm) specified in the Bluetooth standard [1].

One solution of using BLE to send audio data is leveraging the **optional** LE Audio feature introduced in Bluetooth 5.2. However, this does not solve the incompatibility between BLE and Bluetooth-Classic devices. Furthermore, LE Audio does not apply to all BLE devices, and a large portion of BLE devices (i.e., all devices from Bluetooth 4.0 to 5.2) do not support this feature. Even with Bluetooth 5.2 and newer devices, LE Audio and LE isochronous channel features are optional [8], and devices can be qualified for Bluetooth 5.2 without supporting LE Audio [9]. Thus, even devices qualified for Bluetooth 5.2 do not guarantee the use of LE Audio. For example, Apple's latest Airpods do not support LE Audio [10], even though they are qualified for Bluetooth 5.3.

On the headphone side, LE Audio is not just a simple change in the audio application. LE Audio requires a new transport layer (ISOAL) as well as a new proprietary audio codec known as LC3 (due to the throughput limitation of the BLE protocol). This LC3 codec does not exist on most Bluetooth headphones, and using LC3 codec requires licensing through Fraunhofer [11] or licensing through Bluetooth SIG's LE Audio qualification. It is unlikely for chipmakers and headphones manufacturers to pay per-chip LC3 royalties for released products or to go through Bluetooth SIG's qualification process again for LE Audio. Thus, making existing Bluetooth headphones support LE Audio requires more than just a firmware upgrade, and upgrading the firmware also may not be possible for end users.

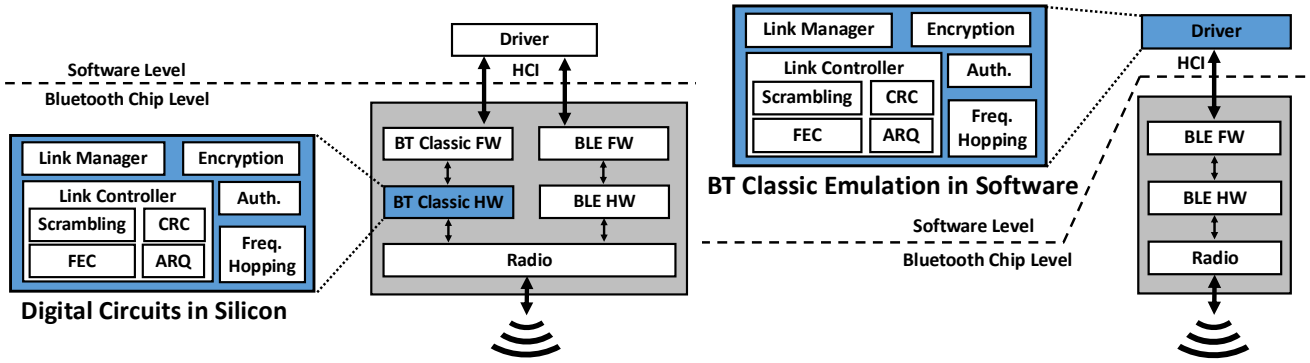
To address this important and practical problem with the majority of headphones using Bluetooth-Classic, we propose BBC, which enables direct communication between BLE and Bluetooth-Classic chips. It emulates full operation of Bluetooth Classic using BLE chips, directly connecting unmodified Bluetooth-Classic headphones with BLE chips.

On the technical front, the key question BBC aims to answer is: *Instead of using dedicated hardware blocks for Bluetooth Classic, can we emulate corresponding blocks in software so that simpler hardware, such as a BLE chip, can appear and function like a Bluetooth-Classic chip?* BBC shows that such software-based emulation is actually possible, even though the operation and processing in Bluetooth Classic are highly hardware-dependent by design. For example, the bit processing, connection establishment, and encryption are directly tied to Bluetooth-Classic's hardware timer. Bluetooth Classic requires timing precisions at the microsecond level, which is challenging for real-time processing in software. Additionally, BBC provides insights into the inner-workings of Bluetooth Classic by presenting a fully-functional, emulation-based Bluetooth system. In the literature, Bluetooth-Classic hardware is treated as black boxes and research on the processing between the raw FSK bits and the HCI layer is lacking. BBC bridges this gap and emulates every step between the two layers.

Benefits. BBC offers several important benefits. First, it is an ideal solution for backward compatibility on newer or BLE-only chips without the need for "dual-mode" chipsets that require two full stacks [5–7]. BBC is especially attractive during the transition from Bluetooth Classic to BLE when billions of headphones are still relying on Bluetooth Classic. Second, even without considering this transition, BBC creates new audio features on current BLE-only devices (such as BLE beacons or BLE fitness trackers) by enabling a new mode of communication. Compared to streaming via LE Audio, BBC does not require any modification to Bluetooth headphones and does not require upgrading the firmware of Bluetooth headphones. Finally, BBC provides the same audio quality as Bluetooth Classic, and blind tests [12, 13] have shown that high-quality audio codecs (AAC or aptX) used in Bluetooth Classic offer noticeably better sound quality than LE Audio, since the LC3 codec uses a very high compression ratio. So, BBC combines the benefits of using simple radio hardware and supporting high audio quality.

For hardware vendors, BBC enables using simpler, cheaper and smaller BLE chips to support both BLE and Bluetooth Classic. BBC directly reduces the cost of future devices, making the solution particularly attractive. It removes hardware complexity using software. Because modern devices have considerably higher computational power than 20 years ago, this emulation only incurs very minimal overhead on modern processors. Our evaluation shows that BBC only requires about 1.6% of the CPU time. For users, BBC solves the problem of incompatibility between different Bluetooth standards without modifications to the hardware. Furthermore, BBC enables new applications that are not possible on existing hardware. For example, audio streaming becomes possible with existing BLE devices using BBC.

To enable Bluetooth-Classic operations without Bluetooth-Classic chips, we propose a new architecture where lower layers of Bluetooth Classic are emulated in the driver. This architecture offers several significant advantages. First, since the driver in BBC directly sends and receives FSK bits, it has complete knowledge of radio transmissions and receptions, and has much better control and diagnostic capabilities. In the traditional Bluetooth architecture, the driver can only send high-level commands and logical data packets whereas the Bluetooth chip applies physical-layer processing, autonomously schedules transmissions and maintains internal connection states. The indirection and complexity make diagnostics difficult, and are prone to incompatibility between different silicon implementations. For example, when Bluetooth does not work, failing to make an initial connection between two Bluetooth devices is a major source of the problem. On traditional systems, these lower-layer operations are autonomously handled and scheduled by the hardware chip, and the driver has limited ability to alter packet exchanges or pinpoint the error. In contrast, BBC is similar to WiFi systems and provides direct visibility of over-the-air transmissions and receptions to the driver. Raw radio bits can be directly monitored to determine the error sources. Second, because the lower-layer processing



(a) A typical Bluetooth dual-mode chip [5–7]

(b) BBC enables Bluetooth Classic on BLE hardware

Figure 2: BBC moves entire Bluetooth-Classic processing from HW to SW domain, enabling BLE to support Bluetooth Classic.

is emulated in the driver, BBC does not require Bluetooth-Classic-specific blocks and components on the radio chip. Thus, BBC can use simple BLE chips as the hardware. Finally, BBC is highly flexible as the lower-layer processing is easily upgradable by compiling new drivers. In contrast, these computations are hardcoded in digital circuits on conventional Bluetooth chips, making it infeasible to upgrade them via software.

BBC matches the operational range of Bluetooth-Classic devices with the same transmit power. Using BLE chips with a transmit power of 4dBm, BBC can stream music without glitching at 10m, which is the same operational range of Class 2 (4dBm [1]) Bluetooth-Classic devices. Note that the advertised range of 100m of Bluetooth requires using "Class 1" devices, which have a transmit power of 20dBm. However, Class 1 devices are for industrial applications and most consumer electronic devices are Class 2 or 3 [14]. BBC simply matches the operational range of Bluetooth Classic with the same transmit power, and does not increase or decrease the communication range. Similar to the case of industrial applications, a higher transmit power (e.g., using power amplifiers) can be used if a greater range is desired.

Challenges. Emulating Bluetooth-Classic's operation is very challenging as its lower layers are tightly coupled with hardware. Furthermore, they consist of numerous highly-intricate components, and implementing the full emulation is highly error-prone. To the best of our knowledge, BBC is the first end-to-end, bidirectional, fully functional Bluetooth-Classic system built from the ground up in academia. BBC is unique in that it accurately emulates various Bluetooth-Classic components (especially below the HCI layer) and enables off-the-shelf BLE chips to establish, negotiate, authenticate, encrypt Bluetooth-Classic connections and then stream audio to Bluetooth headphones, all without the help of any Bluetooth-Classic chip.

Although the design of Bluetooth is documented in the Bluetooth Core Specification [1], it is challenging to build several key hardware components in software and emulate them accurately. One of the challenges is parsing the unconventional use of terminologies in the standard. For example, "baseband" in the Bluetooth standard includes controlling carrier frequencies, connection establishment between two devices, connection authentication, payload encryption, packet ACK/retransmission, addressing and flow controls. These components are not considered as baseband in other (WiFi or cellular) standards. Another challenge is that the frequencies are constantly changing in Bluetooth, and hence the frequency hopping and bit processing must both be correct in order to communicate with Bluetooth headphones. Furthermore, generating frequency-hopping sequences is complicated and is both address- and state-dependent. (The sequences are different before and after a connection is established, and are also different for Central and for Peripheral.) The packet generation and processing involve numerous fields (e.g., headers, scrambler seeds, error checking, encryption) that are time-varying (dependent on the hardware clock) and address-dependent. All of these have to be emulated correctly for successful communication. Finally, debugging Bluetooth communication is difficult because every packet uses a different frequency. This is especially challenging for debugging the connection establishment ("paging") process because the frequency-hopping pattern changes at each packet exchange. It is thus almost impossible for a narrow-band sniffer to reliably trace the entire process as such a sniffer cannot simultaneously capture traffic on two different frequency-hopping sequences.

2 System Design

2.1 Primer of Bluetooth Classic

In a Bluetooth connection pair, one device assumes the role of *Central* while the other is *Peripheral*. Bluetooth Classic is a strict time-slot-based protocol and Central assigns the time slots. Central and Peripheral both maintain a Bluetooth clock for precise time-slot communication. The initial synchronization of two clocks is achieved via *paging*, where Central informs Peripheral its clock value in a 6-packet exchange. In each time slot, packets are sent over a different

RF frequency carrier. The RF frequency depends on the Bluetooth clock and address, and thus Central and Peripheral hop to the same RF frequency once synchronized. The Bluetooth standard defines various packet types. BBC sends ID, FHS, POLL, DM1 and DH3 packets. The first three are used in paging while the last three are used after paging. Each type of packet comes with different encodings and number of fields. ID only contains the access code whereas POLL has the access code and header. FHS, DM1 and DH3 further contain the payload field. Forward-error correction (FEC) is applied to the payload of FHS and DM1. DM1 and DH3 are used for general data communication. DH3 uses up to 3 time slots and has higher throughputs than DM1.

As shown in Fig. 2a, Bluetooth-Classic chips perform a significant amount of bit processing for each packet, which is done by a hardware component called the *Link Controller* (LC). The LC's bit processing varies with time since the computations depend on the hardware clock. To ensure reliable communication, the LC also handles the ARQ (automatic repeat request). The component above the LC is the Link Manager (LM), which runs the Link Manager Protocol (LMP) to establish, negotiate and control a logical Bluetooth connection. The LM also controls link authentication and encryption, implemented in hardware on typical Bluetooth-Classic chips.

2.2 Architecture

The architecture of BBC is shown in Fig. 2b. BBC reuses the hardware essential on BLE chips and emulates all Bluetooth-Classic operations and processing in the driver. BBC generates and processes raw FSK bitstreams in software so that BLE chips can transmit and receive standard Bluetooth-Classic packets. BBC emulates the standard paging process of Bluetooth Classic. After paging, BBC enables slot-based communication with precise timing while also allowing software-based (as opposed to hardware-based) computation. BBC provides reliable communication using retransmission with software ARQ. BBC also emulates all authentication and encryption functions in software. Finally, we complete our Bluetooth system with software-based LM and transport-layer support.

2.3 Connection Establishment

Before streaming audio/music from Central to Peripheral can start, the first step is to establish an initial connection. This is known as *paging* in the Bluetooth standard. The purpose of paging is to synchronize the Bluetooth timers/clocks of two devices. Since Bluetooth is a strict time-slot-based protocol and each time slot uses a different frequency, this process is critical in ensuring that two devices are synchronized in time and follow the same hopping sequence after paging.

On conventional Bluetooth-Classic chips, paging is entirely and autonomously handled by hardware, responsible for all packet/bit processing, Tx/Rx timing, and hop generation. Since BLE chips do not come with these hardware blocks at all, BBC must emulate all these functions in software while utilizing only the existing hardware components on BLE chips: the FSK radio and the timer.

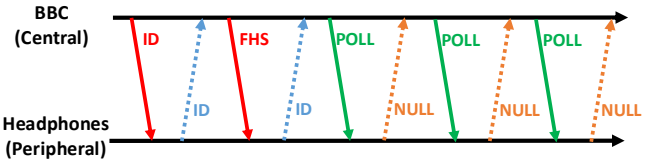


Figure 3: Connection Establishment: BBC always sends all packets whereas ID and NULL may not be received. Paging is successful if ID and NULL are received. The hopping sequence and access code change between ID and POLL.

In the Bluetooth standard, paging is a 6-packet process (and each packet occupies a time slot). Central sends ID, FHS, and POLL packets in odd time slots whereas Peripheral sends ID, ID, and NULL packets in even time slots. The FHS packet contains Central's clock value and thus Peripheral can start following Central's hopping sequence after the POLL packet. By design, paging is highly dependent on hardware for correct processing, timing, and frequency hopping. Note that the frequency hop and the bit processing of FHS, POLL, and NULL packets are time-variant since the value of the hardware clock changes the processing steps. The same logical packet will have random FSK bits in different time slots.

Since Central and Peripheral initially operate with different clocks during paging, the 6-packet process normally fails at different steps. On conventional chips, hardware blocks will then be used to generate new random bits and hops based on the current clock. However, this poses a significant challenge to BBC since BLE chips do not have such hardware for on-the-fly processing.

BBC's paging is based on our key insight: Instead of requiring hardware to generate bits and hops from the Bluetooth clock in real time, the entire paging process can be made deterministic, thus allowing simple BLE chips to page Bluetooth headphones with pre-generated FSK bits and hops. More specifically, the first 4 packets (and the hopping sequence) depend on the Peripheral clock, Peripheral address and a 5-bit number X. In the standard, Central simply uses part of its own clock as a "guess" of X. (The true X is a constantly changing number only known to Peripheral.) Furthermore, the address is known to Central, and the Peripheral clock during paging is only used for alternating Tx/Rx and is always the same pattern. Therefore, the first 4 packets are deterministic if the same clock is used each time. Moreover, the last 2 packets (and the hopping sequence) depend on Central's clock and address, which are specified in the FHS packet. Since we can make the first 4 packets deterministic and have complete control of the FHS packet, the entire paging process is deterministic and can be pre-generated.[2] Finally, since the pre-generated bits and

[2]Bluetooth Classic was designed when the strict FHSS regulation was in

hops are legitimate sequences that conventional chips also generate (when the same X is used), the design of Bluetooth ensures that the Peripheral will be paged. The only difference is that BBC chooses a fixed (instead of a random) X.

In BBC, we pre-generate the FSK bits and hops, and send the raw bits using BLE radio. Each transmission is scheduled using the timer on BLE chips. If the BLE chip receives responses (ID and NULL packets), paging is successful and BBC starts bidirectional packet communication. Otherwise, BBC continuously retries paging.

In the Bluetooth standard, the Peripheral should always respond to the POLL packet with a NULL packet. However, when experimenting with Bluetooth headphones, we find that some headphones do not strictly follow this. To handle this problem, BBC further sends two additional POLLs in the next two transmit slots. We find Apple's headphones usually do not respond to the first POLL and two additional POLLs are particularly crucial.

2.4 Bidirectional Packet Communication

Upon successful paging, BBC emulates the bidirectional packet communication of Bluetooth Classic using BLE chips. We divide the design into two parts. The first part (Sec. 2.4.1) solves packet-level challenges where BBC enables BLE chips to transmit/receive packets in strict time slots and follow the frequency hopping sequence of Bluetooth Classic without dedicated hardware. The second part (Sec. 2.4.2–2.4.3) addresses the bit-level challenges where BBC emulates all bit processing of Bluetooth Classic, including both transmission and reception, in software.

2.4.1 Packet-level Design

Bluetooth Classic's physical and MAC-layer designs are very hardware-centric, which makes software emulation particularly challenging. On the one hand, Bluetooth devices have to maintain a very strict and precise time base to align transmissions at the start of each time slot, which is best achieved by hardware timers. On the other hand, the timer's instant clock value at the time of transmission/reception is also used in the physical and MAC layer processing of each packet. For silicon implementation, these processing blocks can be directly wired to hardware timers. For software emulation, however, it is much more challenging because the computation is done in software and is much further away from the hardware timer.

We address this challenge with a design of ping-pong communication between the driver and the BLE chip. BBC uses the timer on BLE chips to maintain a stable and precise timing. Every 2.5ms, the BLE chip sends the current clock value and

effect, thus requiring random hopping. Section 15.247 was amended in 2002, which allows digital modulation systems to be used. For example, BLE uses fixed advertising channels, which are also used for connection establishment.

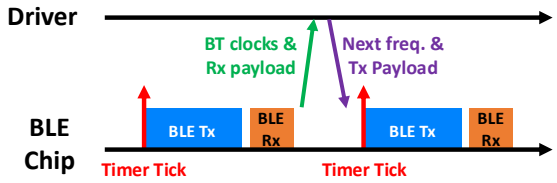


Figure 4: Ping-pong design in BBC to satisfy BT's strict timing

the raw FSK payload to the driver. These pieces of information allow BBC to correctly decode raw bits in software. The BLE chip also sends the clock value of the next transmit slot to the driver. BBC uses this clock to encode the next transmission and calculate the next frequency hops. BBC then sends the next two frequency hops, the number of bytes to be transmitted, and the raw FSK bits to the BLE chip. The bits are not immediately transmitted. Instead, BBC uses the timer to set the frequency hop and always trigger the transmission at the same timer value, which guarantees that each transmission is precisely 2.5ms apart. After each transmission, BBC hops to the new frequency and directly enters the receive mode to collect FSK bits. By decoupling the timer and the computation this way, BBC guarantees precise timing required by the Bluetooth standard while also allowing software-based processing, which naturally has looser timing.

Each time slot in Bluetooth Classic is 625μs and thus one pair of Tx-Rx slots is 1.25ms. However, the standard also defines multi-slot transmissions where a transmission can take 3 slots (followed by 1 slot of reception). We find that this multi-slot transmission is widely used in conventional Bluetooth headphones since it significantly improves throughput. BBC uses an interval of 2.5ms so that both single-slot and multi-slot transmissions are supported. For the (mandatory) single-slot transmission, only 2 slots are used. When streaming audio data, multi-slot transmission is used and all 4 slots are occupied.

BBC configures the length of the transmission when the transmission starts. For the reception, BBC uses a fixed length of 42 bytes after a single-slot transmission and 14 bytes after a multi-slot transmission. This design allows BBC to receive a full packet from Peripheral after a single-slot transmission and to receive the acknowledgement bit after a multi-slot transmission.

2.4.2 Bit-level Design (Transmission)

Since BLE chips do not have the hardware bit-processing blocks for Bluetooth-Classic packets, BBC has to emulate all these blocks in software. Specifically, BBC generates access codes, header/HEC, and payload/CRC. Then, it emulates (time-variant) bit scrambling and FEC coding so that legitimate Bluetooth-Classic packets will be transmitted when the bits are transmitted by BLE radio.

Access Code. Each Bluetooth packet begins with a 72-bit access code. BBC generates the access code in software by applying two XOR (exclusive OR) operations with BCH

(Bose-Chaudhuri-Hocquenghem) coding applied in between.

The packet structures of Bluetooth-Classic and BLE packets are very different. Bluetooth Classic uses 72-bit access codes whereas BLE uses 4-byte access addresses. BBC resolves this difference by setting the access address of BLE to the byte 0 to byte 3 of the generated access codes. This ensures that BLE starts receiving bits when a Bluetooth-Classic packet arrives, and correctly transmits access code bits in Bluetooth Classic's format.

Since two access codes are used during the paging process of Bluetooth Classic, BBC switches access addresses at different slots. Specifically, BBC uses Peripheral's access code when transmitting ID and FHS packets, and uses Central's access code when transmitting POLL packets. If paging is successful, the Central's access code will be used for subsequent traffic.

Header and HEC. Most Bluetooth packets contain an 18-bit header, which consists of 10 bits of information and 8 bits of Header Error Check (HEC). The 10 bits include the packet type (e.g., DM1), a unique address (AM_ADDR), 1 bit (ARQN) for acknowledgement, and 1 bit (SEQN) for data toggling, and BBC fills each field accordingly. BBC specifies AM_ADDR explicitly in the FHS packet and all subsequent traffic contains the same AM_ADDR. The ARQN and SEQN bits are used for the packet retransmission logic and are calculated in Sec. 2.5. To generate the HEC, we build a linear feedback shift register (LFSR) in software and calculate the 10 bits of HEC by the shift register.

Payload and CRC. For data-bearing packets, payload follows the header. In addition to actual data, the payload field contains a separate header known as the *payload header*. The payload header contains LLID and the length of the remaining payload.

The LLID is important for managing fragmentation and LMP. BBC sets LLID to 2 to indicate the start of a logical packet, and to 1 to indicate a continuation fragment. LMP packets have an LLID of 3.

Similar to HEC, BBC runs a software-based LFSR to generate the CRC. The LFSR is also initialized with addresses but has a different length and feedback polynomial. BBC appends the CRC bytes at the end of the packet.

Scrambling. After BBC generates the header and payload (including CRC) in the driver, the entire bitstream is processed by the scrambler. BBC uses a 6-bit LFSR as the scrambler, initialized by bit 6 to bit 1 of the Bluetooth clock. Then, BBC continuously XORs each bit with the LFSR output.

Forward Error Correction (FEC). After scrambling, BBC applies FEC to the bitstream. The header and the payload use different FEC codes. The header portion (the first 18 bits) is encoded with the (3,1) repetition code. For packet types such as FHS and DM1, the payload is encoded with a BCH encoder. For the encoder, BBC uses software LFSR to generate 15 coded bits for every 10 information bits. The input to the encoder is padded so that its length is a multiple of 10. For other packet types, the scrambled payload is directly used.

Pack FSK Bits as BLE Payload. After the FEC, BBC prepends the bitstream with the last 5 bytes of the access code. Since BLE's access address is set to the first 4 bytes of the access code, the first 4 bytes are automatically transmitted by the chip. The entire bitstream is packed as the BLE payload and is sent to the chip for transmission.

2.4.3 Bit-level Design (Reception)

When a Bluetooth-Classic packet arrives, its access code will match BLE's access address and raw FSK bits will be collected. These raw bits, along with the two clock values, are sent directly to the driver for decoding. The decoding process is the reverse process in Sec. 2.4.2, including removing the FEC, descrambling the entire bitstream, checking the HEC and CRC, parsing the fields, and collecting the payload.

When the driver receives the bits from the BLE chip, it first removes the first 14 bytes (which includes two clock values, one byte of total length, and 5 bytes of the tail of the access code). The next 54 bits (which use the repetition code) are decoded using majority vote. The next 255 (=15·17) bits correspond to the payload portion of a packet. We extract the first 10 bits of every 15 bits and concatenate all output bits. Note that DM1 is the only data-bearing packet type that is mandatory for all Bluetooth devices, and the maximum payload length of a DM1 packet is 17 bytes.

The entire bitstream is then descrambled using the clock value of the receive slot. BBC follows a strict processing order (FEC, descrambling, then parsing the fields) that the Bluetooth standard mandates.

After descrambling, BBC checks the HEC by comparing the calculated and received HEC. If the header check passes, BBC further parses the information bits (e.g., the acknowledgement bit) and the packet type. If the incoming packet is DM1, BBC decodes the length of the payload (via the first byte of the payload) and calculates the CRC of the packet. If the CRC passes, BBC checks the LLID. If the LLID is 3, the payload is processed by BBC's emulated LM (Sec. 2.7.1). If the LLID is 2, BBC starts collecting a new logical packet. Such a packet can be larger than the maximum size of DM1, and continuation fragments (LLID=1) may follow. To ensure correct fragmentation, we first calculate the total length of the logical packet when a DM1 with LLID=2 is received (since the length of the logical packet is encoded in the first two bytes of the actual payload). BBC then continuously concatenates all subsequent payload bytes until the total length is reached.

2.5 Reliable Packet Communication

In Sec. 2.4, both transmission and reception are realized with BLE chips. However, the packet delivery is not reliable since packets may be lost or received with error. To provide reliable communication, BBC runs software-based error checking and retransmission. In particular, BBC handles the SEQN (se-

quence number) and ARQN (acknowledgement) bits of each packet in the driver.

We build a software circular FIFO to queue outgoing messages. After a successful paging, the read and write pointers of the FIFO are reset. Each entry in the FIFO contains a pointer to the payload and a flag. The flag indicates whether the payload is an LMP message, whether the payload should use multi-slot transmission, and whether the payload is a start of a logical packet. When the upper layer sends a large packet to BBC, BBC first fragments the packet into a start fragment and continuation fragments to make each fragment smaller than the maximum payload size. All fragments are queued in the FIFO.

At every available transmit slot, BBC checks the FIFO. If the FIFO is not empty, BBC generates the raw bits (Sec. 2.4.2) and transmits them. Otherwise, BBC sends a POLL packet.

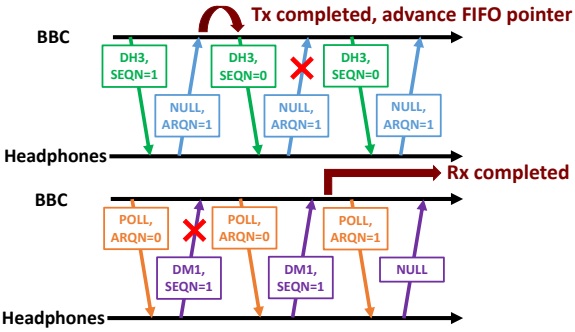


Figure 5: Reliable Transmission and Reception

2.5.1 Processing SEQN and ARQN.

After sending DM1 or DH3, Peripheral should always respond with a packet with a header (such as DM1 or NULL). This header contains the ARQN bit, indicating whether the last transmission is successful. If ARQN is set, BBC advances the read pointer of the FIFO. Otherwise, the read pointer remains and the next transmit slot retransmits the same payload. In Bluetooth Classic, the SEQN bit is toggled for every new message, which is used to detect duplicate transmissions. When BBC receives a data packet, BBC checks the SEQN. If the SEQN is the same as the last data reception, BBC ignores the current reception and does not send the payload to the upper layer or BBC's Link Manager.

2.5.2 Setting SEQN and ARQN.

When sending a packet, BBC also sets SEQN and ARQN so that reliable communication is achieved. If BBC receives a data packet with a correct CRC, ARQN of the next BBC transmission is set to 1. For SEQN, BBC sets this bit to the inversion of the least-significant bit of the FIFO read index. This design comes from the observation that SEQN should toggle for every new message and this maps exactly to the least-significant bit of the index of a circular FIFO with an even number of elements.

2.6 Authentication and Encryption

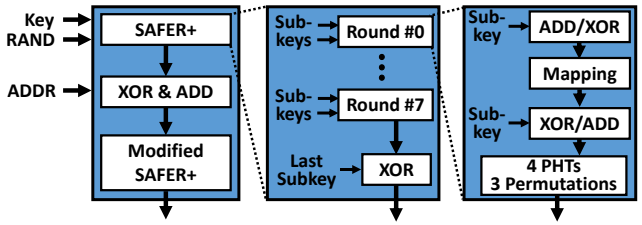


Figure 6: Authentication and Encryption Key Generation

BBC emulates Bluetooth authentication and encryption in software.

2.6.1 Authentication

At the core of Bluetooth authentication is a hash function. This hash function consists of two block ciphers: a SAFER+ cipher and another SAFER+ cipher with a modified third round. The input of the first cipher is a 128-bit key and a 128-bit random number. Within the SAFER+ cipher, the key is first expanded into 17 "subkeys" by applying bit rotation, byte permutation and byte addition. The SAFER+ cipher has 8 rounds and each round uses two subkeys. The input to each round is first added (or XOR'ed) with the first subkey, transformed by a nonlinear mapping, and then added (or XOR'ed) with the second subkey. The bytes are further processed through 4 sets of Pseudo-Hadamard Transform with 3 permutations in between. After 8 rounds, the 17-th subkey is applied to generate the output. The output of the first cipher is then XOR with its input, and combined with the third input to the hash function. The second SAFER+ cipher takes this result as the input and uses an offset key. The final output of the hash function is generated by the second SAFER+ cipher. We implement the entire hash function and the two SAFER+ ciphers in software from the ground up.

2.6.2 Encryption

After authentication, the upper layer starts encryption by issuing HCI_Set_Connection_Encryption. When this command is received, BBC sequentially sends 3 LMP packets (encryption mode, encryption key size, start encryption) to Peripheral. The second LMP packet contains the size of the encryption key and BBC specifies a size of 16 bytes (128 bits).

In Bluetooth, the actual encryption key differs from the link key. The actual key is generated, using the hash function, from the link key, a random number EN_RAND, and ACO (generated during authentication). The 16-byte random number EN_RAND is specified by Central in the third LMP message (LMP_START_ENCRYPTION_REQ). The entire 128-bit output of the hash function is used as the encryption key. BBC reuses the software-based hash function that we built for authentication.

For actual encryption, Bluetooth uses a proprietary algorithm known as E0. We find that emulating E0 in software is

highly intricate. For example, although an open-source implementation of E0 [15] can be found online and although it passes all Bluetooth's test vectors, the implementation is actually incorrect and yields wrong encrypting bitstreams for most Bluetooth addresses. BBC uses our in-house software implementation.

For generating the encryption/decryption sequence, BBC uses 4 software LFSRs with different sizes (25, 31, 33, 39). The LFSRs are initialized with Central's address, current Bluetooth clock, and the 128-bit encryption key. The outputs of the shift registers are fed into an auxiliary logic including a summation and a "blend" logic. The output of E0 is generated by combining (with XOR) the outputs of all shift registers and the auxiliary logic.

In Bluetooth, only the payload portion of a packet is encrypted. E0 is applied after the payload is generated but before scrambling. We add the function call to E0 at the corresponding locations in BBC.

We also design an optimization to minimize the E0 computation during packet transmission. (After various negotiations, the majority of traffic is BBC-to-headphones and thus transmission optimization is the most effective.) Note that the pseudorandom sequence can be pre-generated because the address and the encryption key are known and the Bluetooth clock of the next transmit slot can be estimated. After sending a packet to the BLE chip, BBC estimates the next transmit clock value and calculates the pseudorandom sequence. At the time of the next transmission, if the actual clock value matches the estimation, BBC encrypts the payload by simply performing XOR with the pre-generated sequence.

2.7 Link Management and Transport Layer

To make a BLE chip appear like a Bluetooth-Classic chip to the upper software stack, BBC has software-based Link Manager and L2CAP logic.

2.7.1 Link Manager (LM)

The LM is used in Bluetooth-Classic systems to negotiate and control various properties of the physical link. After paging, two Bluetooth devices must first establish an LMP connection so that other Bluetooth protocols and profiles can use the link. Authentication and encryption are also controlled by LMP.

As the LM, BBC queues LMP_VERSION_REQ after a successful paging. This triggers several message exchanges about LMP features and eventually BBC sends LMP_HOST_CONNECTION_REQ and LMP_SETUP_COMPLETE. At this point a very basic LMP connection is established but no link properties have been set. When BBC receives LMP_SETUP_COMPLETE from headphones, it informs the upper layer of a successful LMP connection via an HCI message (HCI_Connection_Complete).

After LMP_SETUP_COMPLETE, Bluetooth headphones start negotiating link capabilities in detail by sending different LMP requests. BBC emulates the LM by responding to various LMP requests according to the Bluetooth standard.

2.7.2 L2CAP

After an LMP connection is established and with the link authenticated and encrypted, the L2CAP (Logical Link Control and Adaptation Layer Protocol) traffic can be transferred using BBC. From this point on, BBC operates just like a conventional Bluetooth-Classic chip, allowing the upper layers (BlueZ and PulseAudio) to send and receive arbitrary L2CAP data.

After being notified that the link is encrypted, the upper layer automatically initiates various protocols over the L2CAP layer. These include SDP (Service Discovery Protocol [1]), AVCTP (Audio/Video Control Transport Protocol [16]) and AVDTP (Audio/Video Distribution Transport Protocol [17]), which are typically handled by BlueZ. These protocols must be connected first before audio streaming can start. Since BBC appears as a normal Bluetooth-Classic chip to BlueZ, BBC is directly compatible and connected to BlueZ. After negotiation of these protocols, BlueZ continuously sends audio packets, encoded by PulseAudio, over the L2CAP layer of BBC.

3 Implementation and Evaluation

We implement BBC on readily-available BLE chips. For evaluation, we first conduct microbenchmarks using the industry-standard Bluetooth measurement tool. We then perform end-to-end evaluation of BBC by directly connecting and streaming music to unmodified, commodity Bluetooth headphones.

We implement BBC on CC2540 BLE chip from Texas Instruments (TI) and use the CC2540EMK-USB development board [18] as the hardware. We use SDCC (Small Device C Compiler) [19] to develop the firmware and CC-DEBUGGER [20] from TI is used to flash the firmware. The CMD_BLE_TX and CMD_BLE_RX commands are used to transmit and receive BLE packets. We also set the frequency deviation to 160kHz using the MDMCTRL0 register. Since modulation of modern BLE chips is done by digital circuits and radio (e.g., FCC) regulations require measurement of the output power of the unmodulated carrier (i.e., zero frequency deviation), the frequency deviation can be configured. We also build a Bluetooth driver (as a Linux kernel module) to implement Bluetooth-Classic operations. Our driver registers a new HCI Bluetooth device when the kernel detects a matching VID and PID. After opening the Bluetooth device, the kernel sends various HCI messages to the driver. Meanwhile, BBC starts paging Bluetooth headphones after initialization. Once the headphones are successfully paged, the ping-pong communication between the BLE chip and the driver begins. Bidirectional reliable communication is then established.

Since BBC emulates LM in the driver, most of the HCI messages are intercepted and directly responded to by the driver. For a small subset of HCI messages (such as the name request, encryption requests and the ACL traffic), the driver constructs LMP or DH3 packets and puts them in the Tx FIFO. In the driver, the key processing steps (Sec. 2.4–2.5 and Sec. 2.6.2) are implemented in the USB receive callback function. For authentication (Sec. 2.6.1), the hash function is called when the driver receives authentication HCI messages, but the LMP exchanges are implemented in the receive callback function.

3.1 Evaluation Setup

We use Ubuntu 20.04 with BlueZ 5.53 and PulseAudio 14.0 to evaluate BBC. The transmit power of the BLE chip is set to 4dBm, which corresponds to Class 2 devices (with an operational range of up to 10m) in the standard [1]. Thus, in both PHY and system evaluations, we test BBC at 1, 5 and 10m.

For microbenchmarks, we use Teledyne LeCroy's FTS4BT [21], the *de facto* industry-standard [22] sniffer. This Bluetooth test equipment allows us to directly capture the over-the-air (OTA) Bluetooth traffic and measure the physical-layer performance, such as packet error rate (PER) and the maximum achievable throughputs.

Table 1: Evaluation with Unmodified Bluetooth Headphones

Headphones	Bluetooth Chip Used
Sennheiser CX150	Qualcomm QCC3024
Sony SBH20	CSR CSR8640
Apple Airpods 2	Apple H1

We also conduct system evaluation where BBC creates a direct connection with off-the-shelf Bluetooth headphones (Table 1), negotiates the capabilities and streams audio, just like a conventional Bluetooth chip. The system evaluation involves all aspects of Bluetooth, including PHY, packet-retransmission (ARQN/SEQN), and all other components (LMP, authentication/encryption, BlueZ and PulseAudio). On Ubuntu 20.04, the audio codec is implemented in PulseAudio and SBC is used in the system evaluation. Our system evaluation uses the default SBC setting (Bitpool: 53, Block:16, Allocation: Loudness, Subbands:8). This is the "high quality" setting defined in the A2DP standard [4], and is commonly the highest SBC setting of Bluetooth headphones. The setting corresponds to a bitrate of 328kbps.

3.2 Microbenchmark

We evaluate the physical-layer performance at 1, 5, and 10m, and we test POLL, DM1 and DH3 packets at each distance. For each test case, we send 4096 packets. We send the maximum amount of payload for each payload-bearing packet (17 bytes for DM1 and 183 bytes for DH3). From the FTS4BT capture, we count the number of correctly received packets (with correct HEC and correct CRC), and calculate the PER and effective throughputs.

Table 2: Packet Error Rate (PER) and Throughputs at the Physical Layer

POLL			
Distance	1m	5m	10m
Correctly Received Packets	4094	4087	4077
Packet Error Rate (%)	0.05	0.22	0.46
Throughputs (kbps)	No payload		
DM1			
Distance	1m	5m	10m
Correctly Received Packets	4094	4075	4074
Packet Error Rate (%)	0.05	0.51	0.54
Throughputs (kbps)	54.37	54.12	54.11
DH3			
Distance	1m	5m	10m
Correctly Received Packets	3965	3954	3897
Packet Error Rate (%)	3.20	3.47	4.86
Throughputs (kbps)	566.87	565.30	557.15

Table 2 shows the results. For POLL and DM1 packets, the PER is very low. At 1m, only 2 of the 4096 packets are not received correctly, corresponding to a PER of 0.05%. The PER steadily increases with distance, but still remains very low at 10m at around 0.5%. Compared to POLL, DM1 has a slightly higher PER because DM1 packets are longer. The PER is noticeably higher for DH3 packets because DH3 packets have 183 bytes of payload (plus payload header and CRC) and are much longer than DM1 and POLL packets. However, BBC still has good performance with a PER of 3–5%.

We also calculate the throughputs achieved. Table 2 shows that DH3's throughputs are much higher than DM1's because of its much larger packets and lower overheads. Thus, even though DH3 is optional, supporting DH3 packets is important for audio streaming because DH3 is about 10x faster than DM1. We calculate the throughputs by the number of bytes (DM1: 17, DH3: 183) that can be actually used by the L2CAP layer, and we do not count the payload header and CRC bytes as usable payloads. DH3 achieves a throughput of $\sim$560kbps, which provides ample headrooms for the bitrate required by audio streaming (Sec. 3.1).

3.3 System Evaluation

In the system evaluation, BBC directly connects to Bluetooth headphones and streams audio. We measure the performance for 1 min after BBC successfully pages the headphones. Since Bluetooth operates strictly based on time slots, 60s corresponds to 96000 slots. As described in Sec. 2.4.1, transmission is scheduled at a 4-slot interval (which maximizes the throughput of DH3). Therefore, BBC transmits exactly 24000 packets within the 1-min duration. At each transmission, BBC either sends POLL or DM1 (or DH3), depending on the status of the Tx FIFO. We gather the statistics of the number of packets sent and acknowledged. A DM1 or DH3 packet is acknowledged if BBC receives a packet with correct HEC and

Table 3: System Performance with Sennheiser CX150

	1m		5m		10m	
	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)
POLL	8714 / 8825	1.26	8472 / 8617	1.68	8276 / 8549	3.19
DM1	18 / 18	0.00	18 / 18	0.00	19 / 19	0.00
DH3	14709 / 15157	2.96	14723 / 15365	4.18	14588 / 15432	5.47
Total	23441 / 24000	2.33	23213 / 24000	3.28	22883 / 24000	4.65
Total Payload	2490987 Bytes		2493419 Bytes		2470472 Bytes	
Throughput (Avg./Peak)	332.13 kbps/ 338.90 kbps		332.46kbps/ 338.54 kbps		329.40 kbps/ 338.19 kbps	

Table 4: System Performance with Sony SBH20

	1m		5m		10m	
	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)
POLL	8797 / 8873	0.86	8564 / 8654	1.04	8563 / 8701	1.59
DM1	17 / 17	0.00	17 / 18	5.56	17 / 17	0.00
DH3	14701 / 15110	2.71	14701 / 15328	4.09	14701 / 15282	3.80
Total	23515 / 24000	2.02	23282 / 24000	2.99	23281 / 24000	3.00
Total Payload	2491546 Bytes		2491546 Bytes		2491546 Bytes	
Throughput (Avg./Peak)	332.21 kbps/ 336.13 kbps		332.21 kbps/ 338.54 kbps		332.21 kbps/ 336.22 kbps	

ARQN. A POLL packet is acknowledged if a reply with a correct HEC is received. (POLL has no effect on ARQN.) We also calculate the number of payload bytes of all acknowledged packets. We then convert this number to average (60s) and peak (1s) throughputs.

Table 3 shows the system performance with Sennheiser CX150 headphones. Similarly to the microbenchmarks, the PER increases with distance, but the overall PER is still relatively low (∼5%) at 10m. From the individual breakdown, we can see that DH3 has a higher PER because of the longer length. Note that there are very few DM1 packets because we only use DM1 for LMP packets and use DH3 for all other L2CAP traffic. (The Bluetooth standard specifies that DM1 should always be used for LMP.) Finally, because the maximum throughput at the physical layer (∼560 kbps) is significantly higher than the application bitrate (∼328 kbps), the total payload delivered and throughputs are very similar across distances (and are governed by the application bitrate). By using ARQN/SEQN, audio packets are reliably transmitted to the headphones and the audio does not have glitches or interruption. Similarly, Table 4 shows the system performance with Sony SBH20 headphones. The overall PER is only 2–3%. Because Bluetooth uses time slots very strictly and the traffic is entirely scheduled by Central, the traffic pattern is highly predictable at very low PER. Note that the number of acknowledged DM1 and DH3 packets is exactly the same, and thus the total payload and average throughputs are the same. However, the peak throughputs have very small variations due to occasional retransmissions.

Table 5: System Performance with Apple Airpods

	1m		5m		10m	
	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)	Ack'd / Sent	PER (%)
POLL	5103 / 6204	17.75	4813 / 6019	20.04	4774 / 5987	20.26
DM1	21 / 27	22.22	21 / 28	25.00	21 / 26	19.23
DH3	13739 / 17769	22.68	13741 / 17953	23.46	13740 / 17987	23.61
Total	18863 / 24000	21.40	18575 / 24000	22.60	18535 / 24000	22.77
Total Payload	2330133 Bytes		2330499 Bytes		2330316 Bytes	
Throughput (Avg./Peak)	310.68 kbps/ 340.00 kbps		310.73 kbps/ 338.54 kbps		310.71 kbps/ 344.39 kbps	

3.3.1 Deep Dive into Airpods' Bluetooth Connection

Table 5 shows the performance with Apple Airpods, and the PER is significantly higher than the other headphones. We have taken a close look at Airpods' Bluetooth connection. We find that the higher PER comes from the design and implementation on the Airpods' side, and *the result is similar to the PER of using an off-the-shelf Bluetooth chip with Airpods*.

Specifically, as a reference of using a standard Bluetooth chip, we use the Qualcomm WCN6856 card (Qualcomm Atheros FastConnect 6900 [23]) on a Windows 10 laptop to connect to Apple Airpods, and capture the OTA traffic using Teledyne LeCroy's FTS4BT. In the packet trace, we counted the number of packets transmitted by WCN6856 and the number of packets not acknowledged by Airpods. (We manually counted the packets within the duration of 300 packets because this specific condition cannot be easily configured in the FTS4BT suite.) Of the 165 packets transmitted by WCN6856, 30 packets are not acknowledged, which corresponds to a PER of $30/165 = 18.18\%$. `BBC`'s PER (see Table 5) is about 3–5% higher than this baseline, which is similar to the results of CX150.

There are several factors that might cause Airpods to only respond to around 80% of the packets. Airpods are known as true wireless headphones where left and right earbuds both have a Bluetooth chip. Since the original Bluetooth standard [24] was designed to communicate with only one Bluetooth chip, the key design principle of true wireless headphones is to make a pair of Bluetooth chips appear as one *logical* chip. The packet capture between WCN6856 and Airpods shows that Airpods use the standard Bluetooth protocol. However, because Airpods actually have two physical Bluetooth chips, Airpods can be temporarily unable to respond

to some incoming packets due to the need for coordination between the chips on two earbuds.

Note, however, because the PHY provides significantly higher throughputs, BBC actually streams audio to Airpods without interruptions or glitches. Another difference between Airpods and other devices is that Airpods start audio streaming after the "connected" chime (about 5s after successful paging). This contributes to the lower number of total payload bytes and average throughputs in Table 5. In the steady state (11∼60s), the average throughputs are 334.62, 334.68 and 334.67 kbps, which are consistent with prior results.

Another important observation from the packet capture of the Qualcomm WCN6856 with Apple Airpods is that the audio configuration and the authentication/encryption algorithms are exactly the same as BBC. By default, Qualcomm WCN6856 and Apple Airpods use the high-quality A2DP SBC setting (the same setting as Sec. 3.1) and the default authentication (Sec. 2.6.1) and encryption (Sec. 2.6.2). Note that WCN6856 is one of Qualcomm's latest flagship solutions for Bluetooth connectivity on laptops and smartphones, and BBC provides identical audio quality and security for common Bluetooth headphones like Airpods.

3.4 Computation Time

Table 6: Computation Time of BBC

	POLL	DM1	DH3	Encryption
Mean (μs)	1.19	2.32	10.03	29.67
Std. Deviation (μs)	0.65	1.03	2.70	7.62

We evaluate the computation time of BBC using an HP 800 desktop with an i5-4570S processor. BBC only requires simple logic and integer operations, and is efficient on modern hardware. Table 6 shows the time required for generating packets and for precomputing encryption sequences. Packets are sent every 2500μs (4 slots) and the computation (for DH3 packets with encryption) only takes ∼1.6% of the time.

3.5 End-to-end Latency

Table 7: End-to-end Latency

	SBH20	CX150	Airpods
BBC (ms)	161.3	229.9	181.8
CSR dongle (ms)	149.3	216.0	165.3

We measure the end-to-end latency of BBC by collecting the sound samples fed into the Bluetooth codec (via Pulseaudio's sound monitor interface) and the recorded samples of the analog signals from Bluetooth headphones.

Table 7 shows the results. For comparison, we also use a CSR USB dongle and measure the end-to-end latency under identical conditions. In a typical Bluetooth system, the latency is largely contributed by the audio codec. For the mandatory SBC codec, the latency is around 200ms and BBC's latency is consistent with this value. Compared to the USB dongle, the latency difference is only 12.0–16.5ms, which is much less than the audio codec latency. BBC's computation latency (29.67µs) is very small, although the time slot scheduling, packet fragmentation and buffering strategy may affect the latency. We leave these optimizations as future work.

3.6 High-quality Bluetooth Codec

Table 8: BBC with High-quality Audio Codec

	CX150	Airpods	Momentum 2
Codec	AAC	AAC	aptX
PER (%)	6.9	25.2	6.6
Throughput (kbps)	307.40	304.51	352.01

BBC also directly supports high-quality audio codecs by leveraging Bluetooth Classic. We install the `pulseaudio-modules-bt` module [25] on Ubuntu and BBC can directly use these codecs through the same A2DP and L2CAP protocols. CX150 and Airpods support AAC, and we additionally test Sennheiser Momentum 2 with aptX.

The result is provided in Table 8. The codec module by default sets the AAC bitrate to maximum (∼320kbps) whereas aptX has a bitrate of 352kbps. These application-level bitrates are on the same level of SBC's bitrate, and similar to streaming with SBC, BBC can stream audio without glitches. In Table 8, PER is higher than Tables 3–5 due to the noisier environment, but the bitrate targets are still well within the PHY throughput of BBC. In such a scenario, adding the optional adaptive frequency hopping (AFH) feature can be beneficial. Alternatively, we can set AAC's bitrate to 265kbps per Apple's specification [26] and provide more headroom for BBC.

3.7 Comparison

BBC enables using BLE chip for Bluetooth Classic and thus allows smaller chip area. For a fair comparison between BLE and dual-mode chips, we compare CSR's product line, since CSR makes both BLE and dual-mode chips. CSR's smallest BLE chip is CSR1013 with a size of 2.43x2.56mm [27]. In comparison, CSR's smallest dual-mode chips are CSR8610 (3.92x3.68mm [28]) and QCC3026 (3.98x4.02mm [29]), which are 2.3–2.6x the size of CSR1013 with wafer-level packaging. (CC2540 does not use wafer-level packaging and thus the packaged IC is larger. TI's CC2340 is 2.2x2.6mm [30] with wafer-level packaging.) A smaller die size directly affects cost. CSR1013 costs $1.12 [31] whereas QCC3026 costs $5.09 [32]. (CSR8610 is unavailable.) Similarly, CC2540 and CC2340 cost $3.05 [33] and $1.14 [34], respectively, and both are cheaper. In terms of power consumption, CC2540 consumes 15.8mA and 24.6mA during Rx and Tx [35]. Ultra-low-power BLE such as CC2340 consumes 5.3mA (Rx) and 5.1mA (Tx) [30]. In comparison, CSR8640 consumes 15mA [36] during SBC audio streaming. In terms of latency, although the LC3 codec used in LE Audio can reduce the latency to 20∼30ms [37], LE Audio requires support

from both the transmitter and receiver. In comparison, BBC can directly work with existing headphones without requiring this optional feature, and our main focus is addressing the compatibility between BLE and existing headphones. BBC does not require firmware upgrade or any modification to Bluetooth headphones. In Table 1, none of the headphones supports LE Audio. Furthermore, BBC and LE Audio are not orthogonal and can complement each other. BBC can be used as a compatibility mode on BLE chips to support Bluetooth Classic headphones.

4 Discussion

4.1 Using BBC with other BLE chips

BBC only requires BLE chips to have a hardware timer and modulation control, and is portable to other BLE chips. The BLE standard requires maintaining a microsecond-level timer to precisely follow the T_IFS (Inter Frame Space), and this can be directly reused by BBC for Bluetooth Classic. Also, the timer does not have to be hard-wired to BLE's digital circuits and it can just be a general-purpose timer accessible to microcontrollers. Even if there is no timer peripheral (unlikely on modern microcontrollers), ARM core has a built-in system timer (SysTick) as part of the ARM design.

In terms of modulation control, the frequency deviation of common BLE chips is digitally configurable and supports a frequency deviation of around 160kHz. For example, Nordic's chips can use a frequency deviation of 170kHz [38,39], which is within the specification (140 to 175kHz) of Bluetooth Classic. Another example is that the frequency deviation of Silicon Labs' EFR32 chips is configurable [40] between 0 to 1000kHz [41]. These configurations exist because Nordic's legacy protocol (known as ShockBurst) has a standard frequency deviation of 160kHz, and other vendors include this configuration to enable communication with a large number of Nordic's devices. BBC can reuse this property for Bluetooth Classic communication.

5 Related Work

Few academic or open-source research investigates the internals (between FSK and HCI layers) of Bluetooth, and BBC is the only fully operational Bluetooth system (from connection establishment to audio streaming) to the best of our knowledge. Some prior work, with the bottom-up approach, focuses on sniffing Bluetooth traffic. Ubertooth [42] can follow and passively sniff an active Bluetooth connection, and is used in numerous projects [43–45]. However, it operates on the raw FSK level (without Bluetooth bit processing or packet processing) and lacks the ability to create (i.e., paging), maintain (ARQN and LMP), or encrypt a connection. BlueEar [46] adds Bluetooth clock acquisition and adaptive hop selection to Ubertooth for sniffing. Sniffing raw FSK bits using USRP [46,47] is also proposed.

Some other work, with the top-down approach, focuses on leveraging hidden features on conventional Bluetooth chips for low-level Bluetooth experiments. InternalBlue [48–50] reverse-engineers Broadcom's Bluetooth chips and directly sends LMP messages below the HCI layer. BrakTooth Sniffer [51] reverse-engineers the ESP32's Bluetooth implementation, and allows sniffing or injecting of Bluetooth packets. Their goal is modifying the normal packet exchange on Bluetooth-Classic chips, whereas BBC aims to emulate full Bluetooth Classic with BLE chips. They focus on packet-level (POLL, LMP, etc.) interactions in the connected state, whereas BBC handles bit-level processing, paging, frequency hopping and reliable delivery.

Some researchers investigate the security of Bluetooth. SoK [52] provides an overview of the security flaws in Bluetooth. An open-source (but incorrect) E0 implementation [15] has been attempted. KNOB [53,54] and BIAS [55,56] focus on the authentication and key entropy in Bluetooth. Blacktooth [57] implements more attacks using InternalBlue and BIAS. BLUR [58,59] uncovers the flaws of key derivation across Bluetooth and BLE. BrakTooth [60] discovers abnormal Bluetooth packets that trigger crashes and deadlocks.

BBC is also related to CTC research. BlueFi [61] and WiBeacon [62] enable communication from WiFi to Bluetooth. They are limited to either broadcasts or one-way communication for audio streaming (i.e., without connection establishment). FLEW [63] and Unify [64] convert Bluetooth chips into WiFi chips. Bluetooth-Zigbee communication [65,66] is also possible, but the research focuses on signal processing of waveforms and not on full protocol and system-level emulation.

6 Conclusion

BBC is a complete system that enables BLE chips to directly support Bluetooth Classic. By eliminating the need for Bluetooth-Classic hardware blocks, BBC enables using simpler hardware (BLE chips) in the future while maintaining compatibility in the driver. BBC also enables existing BLE chips to directly communicate with Bluetooth headphones, opening the possibility of new audio applications on current BLE devices. Our extensive evaluations demonstrate BBC's low PER and high throughputs, and its ability to stream audio to conventional headphones with the same audio quality as COTS Bluetooth-Classic chips.

References

[1] Bluetooth SIG. Bluetooth Core Specification v5.3. https://www.bluetooth.com/specifications/specs/core-specification-5-3/, 2021.

[2] Bluetooth SIG. 2024 Bluetooth Market Update. https://www.bluetooth.com/2024-market-update/, 2024.

[3] Bluetooth SIG. 2023 Bluetooth Market Update. https://www.bluetooth.com/2023-market-update/, 2023.

[4] Bluetooth SIG. Advanced Audio Distribution Profile 1.4. https://www.bluetooth.com/specifications/specs/advanced-audio-distribution-profile-1-4/, 2022.

[5] MathWorks. Bluetooth Protocol Stack. https://www.mathworks.com/help/bluetooth/ug/bluetooth-protocol-stack.html, 2024.

[6] Kevin Townsend, Carles Cuf, Akiba, and Robert Davidson. *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. O'Reilly Media, Inc., 1st edition, 2014.

[7] Design and Reuse. Bluetooth Dual Mode v5.3 Baseband Controller IP. https://www.design-reuse.com/sip/bluetooth-dual-mode-v5-3-baseband-controller-ip-ip-48804/, 2024.

[8] Intel Corporation. Intel Bluetooth: Isochronous channels and other mandatory features. https://community.intel.com/t5/Wireless/Intel-Bluetooth-Isochronous-channels-and-other-mandatory/m-p/1595248#M53835, 2024.

[9] Ezurio. Is LE Audio mandatory with BT5.2? https://www.ezurio.com/support/faqs/le-audio-mandatory-bt52?srsltid=AfmBOorGyIEkfekvshpCrxIFj7q2gKmbP4By1K-BI3es4ACAw8E6FEVw, 2025.

[10] Simon Cohen. Apple AirPods 4 review: a bit better than basic. https://www.digitaltrends.com/home-theater/apple-airpods-4-review/, 2024.

[11] Fraunhofer IIS. Fraunhofer IIS licenses LC3 audio codec software to Microsoft. https://www.audioblog.iis.fraunhofer.com/fraunhoferiis-lc3-microsoft, 2020.

[12] Google. Low Complexity Communication Codec (LC3). https://github.com/google/liblc3, 2022.

[13] Personal blind comparison of the Bluetooth codecs, AAC vs LC3, re-encoding. https://hydrogenaud.io/index.php/topic,122575.0.html, 2022.

[14] Adoroma. What is Bluetooth Range? What You Need to Know. https://www.adorama.com/alc/bluetooth-range/, 2022.

[15] Arnaud Delmas. A C implementation of the Bluetooth stream cipher E0. https://github.com/adelmas/e0, 2015.

[16] Bluetooth SIG. A/V Control Transport Protocol 1.4. https://www.bluetooth.com/specifications/specs/a-v-control-transport-protocol-1-4/, 2012.

[17] Bluetooth SIG. A/V Distribution Transport Protocol 1.3. https://www.bluetooth.com/specifications/specs/a-v-distribution-transport-protocol-1-3/, 2012.

[18] Texas Instruments. CC2540EMK-USB. https://www.ti.com/tool/CC2540EMK-USB, 2023.

[19] Sandeep Dutta. SDCC - Small Device C Compiler. https://sdcc.sourceforge.net/, 2024.

[20] Texas Instruments. CC-DEBUGGER. https://www.ti.com/tool/CC-DEBUGGER, 2014.

[21] Teledyne LeCroy. FTS4BT Bluetooth Protocol Analyzer and Packet Sniffer. https://fte.com/products/FTS4BT.aspx, 2009.

[22] Frontline Test Equipment. Frontline Introduces World's Only Bluetooth v3.0 + HS Protocol Analyzer. https://fte.com/docs/PressReleases/FTS4BT-HS-press-release.pdf, 2009.

[23] Qualcomm. Qualcomm FastConnect 6900 System. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/qualcomm-fastconnect-6900-product-brief_finalv7.pdf, 2020.

[24] Bluetooth SIG. Specification of the Bluetooth System, 1999.

[25] Huang-Huang Bao. pulseaudio-modules-bt. https://github.com/EHfive/pulseaudio-modules-bt, 2020.

[26] Apple Inc. Accessory Design Guidelines for Apple Devices. https://developer.apple.com/accessories/Accessory-Design-Guidelines.pdf, 2023.

[27] Qualcomm. CSR1013. https://www.qualcomm.com/products/technology/bluetooth/csr101x-series/csr1013, 2015.

[28] Qualcomm. CSR8610. https://www.qualcomm.com/products/internet-of-things/consumer/audio/csr8610, 2018.

[29] Qualcomm. QCC3026. https://www.qualcomm.com/products/internet-of-things/consumer/audio/qcc30xx-series/qcc3026, 2018.

[30] Texas Instruments. CC2340R SimpleLink Family of 2.4GHz Wireless MCUs. https://www.ti.com/lit/ds/symlink/cc2340r5.pdf, 2023.

[31] DigiKey Corporation. CSR1013A05-IUUM-R. https://www.digikey.com/en/products/detail/qualcomm/CSR1013A05-IUUM-R/5640481, 2025.

[32] DigiKey Corporation. QCC-3026-0-81WLNSP-TR-00-0. https://www.digikey.com/en/products/detail/qualcomm/QCC-3026-0-81WLNSP-TR-00-0/21722333, 2025.

[33] DigiKey Corporation. CC2540F128RHAR. https://www.digikey.com/en/products/detail/texas-instruments/CC2540F128RHAR/2534018, 2025.

[34] DigiKey Corporation. CC2340R21N0RGER. https://www.digikey.com/en/products/detail/texas-instruments/CC2340R21N0RGER/22539646, 2025.

[35] Texas Instruments. 2.4-GHz Bluetooth low energy System-on-Chip. https://www.ti.com/lit/ds/symlink/cc2540.pdf, 2013.

[36] Lite on Technology Corporation. BT V4.0 LE Dual Mode Bluetooth Stereo Audio Module WB116C CSR 8640. https://fcc.report/FCC-ID/MDZ-WB116C/1998708.pdf, 2013.

[37] Avantree. AS70P - Latency Levels with Different Devices. https://support.avantree.com/hc/en-us/articles/44460172603417-AS70P-Latency-Levels-with-Different-Devices, 2025.

[38] Nordic Semiconductor. nRF52840 Product Specification v1.11. https://docs-be.nordicsemi.com/bundle/ps_nrf52840/attach/nRF52840_PS_v1.11.pdf?_LANG=enus, 2024.

[39] Nordic Semiconductor. nRF5340 Product Specification v1.6. https://docs-be.nordicsemi.com/bundle/ps_nrf5340/page/nRF5340_PS_v1.6.pdf?_LANG=enus, 2025.

[40] Silicon Labs. EFR32xG22 Wireless Gecko Reference Manual. https://www.silabs.com/documents/public/reference-manuals/efr32xg22-rm.pdf, 2024.

[41] Silicon Labs. AN971: EFR32 Radio Configurator Guide for RAIL in Simplicity Studio v4. https://www.silabs.com/documents/public/application-notes/an971-efr32-radio-configurator-guide.pdf, 2025.

[42] Michael Ossmann. GREAT SCOTT GADGETS Ubertooth One. https://greatscottgadgets.com/ubertoothone/, 2020.

[43] Maxim Chernyshev, Craig Valli, and Michael Johnstone. Revisiting urban war nibbling: Mobile passive discovery of classic bluetooth devices using ubertooth one. *Trans. Info. For. Sec.*, 12(7):1625–1636, jul 2017.

[44] Aaron Kinfe, Chijung Jung, Kai Lin, Marshall Clyburn, and Fnu Suya. Hackwrt: Network traffic-based eavesdropping of handwriting. In *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, CPS-IoT Week '23, page 55–60, New York, NY, USA, 2023. Association for Computing Machinery.

[45] Thomas Willingham, Cody Henderson, Blair Kiel, Md Shariful Haque, and Travis Atkison. Testing vulnerabilities in bluetooth low energy. In *Proceedings of the ACMSE 2018 Conference*, ACMSE '18, New York, NY, USA, 2018. Association for Computing Machinery.

[46] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. Practical bluetooth traffic sniffing: Systems and privacy implications. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 333–345, New York, NY, USA, 2016. Association for Computing Machinery.

[47] Marco Cominelli, Francesco Gringoli, Paul Patras, Margus Lind, and Guevara Noubir. Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 534–548, 2020.

[48] Jiska Classen and Matthias Hollick. Inside job: diagnosing bluetooth lower layers using off-the-shelf devices. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, page 186–191, New York, NY, USA, 2019. Association for Computing Machinery.

[49] The InternalBlue Team. Bluetooth experimentation framework for Broadcom and Cypress chips. https://github.com/seemoo-lab/internalblue, 2021.

[50] Dennis Mantz. InternalBlue—A Bluetooth Experimentation Framework Based on Mobile Device Reverse Engineering. Master's thesis. Technische Universität Darmstadt. http://tubiblio.ulb.tu-darmstadt.de/107125/, 2018.

[51] Matheus Eduardo. BrakTooth ESP32 BR/EDR Active Sniffer/Injector. https://github.com/Matheus-Garbelini/esp32_bluetooth_classic_sniffer, 2023.

[52] J. Wu, R. Wu, D. Xu, D. Tian, and A. Bianchi. Sok: The long journey of exploiting and defending the legacy of king harald bluetooth. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 23–23, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[53] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. The knob is broken: Exploiting low entropy in the encryption key negotiation of bluetooth br/edr. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, August 2019.

[54] NIST. CVE-2019-9506. `https://nvd.nist.gov/vuln/detail/CVE-2019-9506`, 2019.

[55] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Bias: Bluetooth impersonation attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2020.

[56] NIST. CVE-2020-10135. `https://nvd.nist.gov/vuln/detail/CVE-2020-10135`, 2020.

[57] Mingrui Ai, Kaiping Xue, Bo Luo, Lutong Chen, Nenghai Yu, Qibin Sun, and Feng Wu. Blacktooth: Breaking through the defense of bluetooth in silence. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 55–68, New York, NY, USA, 2022. Association for Computing Machinery.

[58] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. Blurtooth: Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 196–207, New York, NY, USA, 2022. Association for Computing Machinery.

[59] NIST. CVE-2020-15802. `https://nvd.nist.gov/vuln/detail/CVE-2020-15802`, 2020.

[60] Matheus E. Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. BrakTooth: Causing havoc on bluetooth link manager via directed fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1025–1042, Boston, MA, August 2022. USENIX Association.

[61] Hsun-Wei Cho and Kang G. Shin. Bluefi: bluetooth over wifi. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 475–487, New York, NY, USA, 2021. Association for Computing Machinery.

[62] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. Wibeacon: expanding ble location-based services via wifi. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 83–96, New York, NY, USA, 2021. Association for Computing Machinery.

[63] Hsun-Wei Cho and Kang G. Shin. Flew: fully emulated wifi. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, page 29–41, New York, NY, USA, 2022. Association for Computing Machinery.

[64] Hsun-Wei Cho and Kang G. Shin. *Unify: Turning BLE/FSK SoC into WiFi SoC*. Association for Computing Machinery, New York, NY, USA, 2023.

[65] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. Bluebee: A 10,000x faster cross-technology communication via phy emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.

[66] Wenchao Jiang, Song Min Kim, Zhijun Li, and Tian He. Achieving receiver-side cross-technology communication with cross-decoding. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 639–652, New York, NY, USA, 2018. Association for Computing Machinery.

A Alternative Design Choices

We have explored alternative design options in `BBC`. However, we find that they either make the system incompatible with Bluetooth headphones, or cannot establish a connection at all. In particular, we have tried using the open-source E0 cipher. However, this breaks the compatibility with commodity Bluetooth headphones and cannot stream music at all because the E0 cipher does not produce the correct bit sequence. Also, in the LMP implementation, we tried streaming music without setting up link authentication and encryption first (since they are technically optional according to the standard). However, we find that Bluetooth headphones will not accept audio packets without authentication and encryption. Finally, for the slotted communication, we have tried longer transmission intervals (instead of 2500μs). However, we find that such a design cannot sustain the required throughput for music streaming, which results in audio glitches.

B Initialization Steps of E0

In `BBC`, the shift registers and the auxiliary logic are first run for 240 cycles. During the last 128 cycles, the E0 output is temporarily stored as a variable Z. At the 240-th cycle, the content of the 4 shift registers is instantaneously updated ("parallel load") with Z. At that instant, E0 also starts outputting the pseudorandom sequence to be used for encrypting the payload. However, the auxiliary logic has internal states and they are carefully maintained to generate the correct E0 output. Specifically, `BBC` first pauses the update of the auxiliary logic, loads 4 registers with Z, and generates the first

pseudorandom bit before updating the auxiliary logic again. This order is followed so that the first encryption bit is correct. BBC then keeps on generating the pseudorandom sequence until the end of a packet.

C Encryption Check using CRC

A peculiar design of Bluetooth is that the CRC is encrypted, and BBC leverages this property to detect whether the payload is encrypted or not. Specifically, after BBC sends the third LMP encryption request (LMP_START_ENCRYPTION_REQ), Peripheral should turn on encryption and reply with an encrypted LMP_ACCEPTED packet. When this packet is received, BBC first tries, without decryption, to decode the packet and check the CRC. If the packet is encrypted, the CRC check fails, and BBC tries to decode the same packet with decryption enabled. If the CRC check (after decryption) passes, BBC sets an internal flag (`encryptionenabled`) to 1. If this flag is set, BBC enables encryption for any outgoing payload. After the encrypted LMP_ACCEPTED packet is received, BBC informs the upper layer that encryption is turned on.

This two-step CRC check ensures that BBC is resilient when the encryption exchange is different from the ideal case. For example, Peripheral may be temporarily unable to turn on encryption and send an unencrypted LMP_NOT_ACCEPTED packet. Or, Peripheral sends other unencrypted packets before LMP_START_ENCRYPTION_REQ is processed. BBC's design ensures that these packets can still be received correctly regardless of Peripheral's encryption state.

D Initial LMP Exchanges

When establishing an initial LMP connection, BBC's LMP exchanges are slightly different from the reference message chart in the Bluetooth standard [1]. In the reference chart, after LMP_HOST_CONNECTION_REQ, the Central and Peripheral will negotiate various link capabilities, authenticate and encrypt the link before finally exchanging LMP_SETUP_COMPLETE. However, commercial Bluetooth headphones tend to first establish a very simple connection before negotiating Bluetooth capabilities. We design the LM in BBC as described in Sec. 2.7.1 so that it is directly compatible with COTS headphones.

E HCI Interaction during Authentication

BBC intercepts and emulates the interaction with the upper layer (HCI) during authentication. When BBC receives HCI_Authentication_Requested, it first sends HCI_Link_Key_Request to retrieve the link key from the upper layer. The link key is first input to the hash function. BBC then sends LMP_AU_RAND to Peripheral. The LMP_AU_RAND packet contains a 128-bit random number (AURAND), which is the second input. Finally, the third input to the hash function is Peripheral's address. Peripheral responds to LMP_AU_RAND with LMP_SRES, containing the first 4 bytes of the hash function's output. If LMP_SRES is received, BBC notifies the upper layer that authentication is completed. The remaining 12 bytes of the hash are known as ACO, which is essential for the encryption bit processing.

F Synchronization Procedure for FTS4BT

Because the frequency hopping and bit scrambling depend on the Bluetooth clock, Teledyne LeCroy's FTS4BT tool has to be synchronized with BBC first before capturing the packets. The FTS4BT suite provides three possible clock synchronization methods and we use the "Central Inquiry" mode. In this mode, FTS4BT sends an ID packet and BBC should reply with an FHS packet. Since the FHS packet contains BBC's current clock value, FTS4BT can be synchronized with BBC's clock. We add this synchronization message exchange in firmware to satisfy the synchronization requirement of FTS4BT. This exchange is very similar to the second to third packets of the paging process (Sec. 2.3). The key difference is that instead of using Peripheral's access code, the GIAC (General Inquiry Access Code) and the corresponding hopping sequence should be used. Additionally, HEC and CRC shift registers are initialized with zeros. Following this procedure, once BBC receives ID and sends FHS, interrupts are enabled to start packet transmission, and FTS4BT is now in sync with BBC and can capture packets from BBC.