BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy

Anonymous Submission

***Abstract*—The Bluetooth standard specifies two transports: Bluetooth Classic (BT) for high-throughput wireless services and Bluetooth Low Energy (BLE) for very low-power scenarios. BT and BLE have dedicated pairing protocols and devices have to pair over BT and BLE to use both securely. In 2014, the Bluetooth standard (v4.2) addressed this usability issue by introducing *Cross-Transport Key Derivation (CTKD)*. CTKD allows establishing BT and BLE pairing keys just by pairing over one of the two transports. While CTKD crosses the security boundary between BT and BLE, little is known about the internals of CTKD and its security implications.**

In this work, we present the first complete description of CTKD obtained by merging the scattered information from the Bluetooth standard with the results from our reverse-engineering experiments. Then, we perform a security evaluation of CTKD and uncover four cross-transport issues in its specification. We leverage these issues to design four standard-compliant attacks on CTKD enabling new ways to exploit Bluetooth (e.g., exploiting BT and BLE by targeting only one of the two). Our attacks work even if the strongest security mechanism for BT and BLE are in place, including Numeric Comparison and Secure Connections. They allow to impersonate, man-in-the-middle, and establish unintended sessions with arbitrary devices. We refer to our attacks as *BLUR attacks*, as they *blur* the security boundary between BT and BLE. We provide a low-cost implementation of the BLUR attacks and we successfully evaluate them on 16 devices with 14 unique Bluetooth chips from popular vendors. We discuss the attacks' root causes and present effective countermeasures to fix them. We disclosed our findings and countermeasures to the Bluetooth SIG in May 2020 (CVE-2020-15802), and we reported additional unmitigated issues in May 2021.

***Keywords*—CTKD, Bluetooth, Cross-Transport Attacks**

I. Introduction

Bluetooth is a pervasive wireless technology used by billions of devices including mobile phones, laptops, headphones, cars, speakers, medical, and industrial appliances [11]. It is specified in an open standard maintained by the Bluetooth special interest group (SIG), and its latest version is 5.2 [10]. The standard specifies two transports: *Bluetooth Classic (BT)* and *Bluetooth Low Energy (BLE)*. BT is best suited for connection-oriented and high-throughput use cases, such as streaming audio. BLE is optimized for connection-less and very-low-power use cases such as fitness tracking.

The Bluetooth standard defines dedicated security architectures and threat models for BT [10, p. 947] and BLE [10, p. 1617]. Each transport provides a *pairing* and a *session establishment* protocol. Pairing results in the establishment of a long-term pairing key that acts as the root of trust. Session establishment allows paired devices to establish a secure channel through a fresh session key derived from their shared pairing key.

Traditionally, two devices supporting BT and BLE would have to pair separately on each transport. In 2014, the Bluetooth standard (v4.2) introduced *Cross-Transport Key Derivation (CTKD)* to address this usability issue. CTKD enables to pair devices once, either over BT or BLE, and negotiate BT and BLE pairing keys without having to pair a second time [10, p. 1401].

CTKD has not received any attention from the research community and the Bluetooth standard describes only some aspects and threats associated with CTKD. We believe CTKD provides a significant attack surface, as it is a standard-compliant security feature, used in conjunction with the most secure Bluetooth modes (e.g., Secure Connections), and is transparent to the end-users. In addition, CTKD allows *crossing* the security boundary between BT and BLE as CTKD forces implicit trust between BT and BLE. For example, if two devices pair over BT and generate a BLE pairing key with CTKD then the security of the BLE transport entirely relies on BT.

In this work, we present a complete description of CTKD obtained by combining the incomplete and scattered information provided in the Bluetooth standard and our reverse-engineering experiments needed to understand how CTKD is negotiated and used in practice for BT and BLE. Then, we perform the first security evaluation of CTKD and we uncover four *cross-transport issues (CTI)* with its specification. The issues affect pairing states, role asymmetries, key generation and distribution, and association methods. For example, CTKD by-design enables overwriting trusted keys with malicious ones across transports.

We leverage the uncovered CTIs to design four *cross-transport* and *standard-compliant* attacks on CTKD. Our attacks enable persistent cross-transport impersonation, man-in-the-middle, and unintended session attacks on BT and BLE via CTKD. The attacks are effective regardless of the employed Bluetooth security mechanisms, including Secure Connections and Numeric Comparison. We name our attacks *BLUR attacks*, as theyblur the security boundary between BT and BLE.

The BLUR attacks are the first standard-compliant BT and BLE attacks to not require the attacker to be present when a victim is pairing or establishing a secure session, unlike prior work [22], [21], [36], [40], [9], [2], [5], [4], [44], [46], [41]. In particular, our attacks are the first that can be conducted in absence of one of the victims. The BLUR attacks are also the first that exploit interactions between BT and BLE (via CTKD). For a more detailed comparison to prior work see Section VIII.

To demonstrate that the BLUR attacks are feasible we present a low-cost implementation of the attacks based on a cheap development board and open-source software. We evaluated our attacks on a large and heterogeneous sample of devices. In particular, we exploited *16* unique devices employing *14* different Bluetooth chips from Broadcom, Cambridge Silicon Radio (CSR), Cypress, Intel, and Qualcomm. Our set of vulnerable devices covers *all* Bluetooth versions supporting CTKD (i.e., Bluetooth 4.2, 5.0, 5.1, and 5.2) and even a 4.1 device to which CTKD was backported.

We concretely address the BLUR attacks by presenting four *protocol-level* countermeasures mitigating the presented CTIs and the BLUR attacks. Our mitigations can be implemented at the operating system level with low effort. To backup this claim we tested one countermeasure (i.e., disable key overwriting) by implementing it on a Linux laptop.

We responsibly disclosed our findings with the Bluetooth SIG two times. In May 2020 we sent our first report which was tracked with CVE-2020-15802. In September 2020 the Bluetooth SIG unilaterally released a security note (see https://tinyurl.com/vxhwftc2), claiming that Bluetooth 5.1 and later are not vulnerable to the presented attacks. As result, we further analyzed 5.1 and 5.2 devices, and found them to still be susceptible. We explain why this is the case in Section IV-F and experimentally confirm it in Section VI-B. We disclosed those findings to the SIG, but have not received a reaction. We note that the SIG is expected to notify vendors of vulnerabilities, so no separate vendor disclosure is required.

We summarize our main contributions as follows:

- We present a complete description of CTKD combining public and reverse-engineered information. We perform the first security evaluation of CTKD and uncover four vulnerabilities in its specification. For example, CTKD enables to adversarially pair over unused transports and to tamper with BT and BLE security keys.
- Based on the identified issues we propose four novel and standard-compliant attacks capable of breaking BT and BLE just by targeting one of the two. Compared to related work, our attacks are the first exploiting CTKD and acting across transports. Our attacks enable to impersonate, man-in-the-middle, and establish unwanted and stealthy sessions with arbitrary devices. We name our attacks *BLUR attacks* as they blur the security boundary between BT and BLE.
- We present a low-cost implementation of the BLUR attacks based on a Linux laptop and a Bluetooth development board. We use our implementation to attack 16 different devices employing 14 unique Bluetooth chips and covering all Bluetooth versions compatible with CTKD (e.g., 4.2, 5.0, 5.1, and 5.2). Our evaluation demonstrates that the BLUR attacks are very effective and specification-compliant. To address them, we discuss four countermeasures to address the presented issues and attacks affecting CTKD.

II. Bluetooth Classic (BT) and Low Energy (BLE)

BT and BLE are two wireless transports specified in the Bluetooth standard [10]. These transports are designed to complement each other. BT is used for high-throughput and connection-oriented services, such as streaming audio and voice, while BLE is optimized for very low-power and low-throughput services such as fitness tracking and digital contact tracing. High-end devices, such as laptops, smartphones, headsets, and tablets, provide both BT and BLE, while low-end devices such as mice, keyboards, and wearables provide either BT or BLE.

BT and BLE have similar security mechanisms (i.e., pairing and session establishment) but *different* security architectures and threat models. Pairing, also known as Secure Simple Pairing (SSP), lets two devices establish and authenticate a pairing key that acts as the root of trust. BLE SSP is performed over the Security Manager Protocol (SMP) [10, p. 1666], while BT SSP uses the Link Manager Protocol (LMP) [10, p. 568]. During pairing, BLE allows negotiating the entropy of the pairing key while BT does not.

While pairing, BT and BLE employ similar *association mechanisms*. For example, *Just Works* association is supported by all Bluetooth devices as it does not require user interaction, but it does not protect against MitM attacks. While *Numeric Comparison* association protects against MitM attacks by asking the user to confirm a numeric code on the pairing devices' screens. As the Bluetooth standard does not protect the negotiation of the association method, an attacker can always *downgrade* it to Just Works even if the victim device has I/O capabilities.

Session establishment lets paired devices establish a secure communication channel. The channel is protected by a fresh session key derived from the pairing key and some nonces. During session establishment, BT allows negotiating the entropy of the session key while the BLE session key inherits the entropy of the associated pairing key.

BT and BLE use the same notion of *pairable* and *discoverable* states. If a device is pairable then it accepts pairing requests from other devices. If it is discoverable it reveals its identity when scanned by other devices. Notably, a device answers to a pairing request even if it is *not* discoverable [43]. For example, if the user knows the Bluetooth address of her pair of headphones she can complete BT or BLE pairing by sending a pairing request from her laptop without putting the headphones into discoverable mode.

BT and BLE provide a *Secure Connections* mode which enhances the security primitives in use without affecting the underlying protocols. In particular, Secure Connections mandates the usage of FIPS-compliant algorithms such as AES-CCM, HMAC-SHA-256, and the ECDH on the P-256 curve [10, p. 269].

Both BT and BLE use a *master-slave* medium access protocol. The master (BLE central) is the connection initiator, while the slave (BLE peripheral) is the responder. BT allows to switch roles dynamically, while BLE roles are fixed. High-end devices, such as laptops and smartphones, support both BLE master and BLE slave modes and are typically used as BLE masters, while low-end devices, such as fitness trackers and smartwatches support only the BLE slave mode [1].

[1]For precise technical descriptions in the rest of the paper we follow the *Bluetooth standard*'s master/slave terminology instead of more apt terms like leader/follower.

III. Description and Security Analysis of CTKD

In this section, we present the first complete description and security analysis of CTKD. In Section III-A we describe what is publicly known about CTKD, in Section III-B we complement those information with other crucial ones that we had to reverse-engineer (e.g., CTKD negotiation for BT and BLE). In Section III-C, we uncover four critical and novel *cross-transport issues (CTI)* in the specification of CTKD. These issues are the root causes of the standard-compliant and cross-transport attacks presented in Section V and evaluated in Section VI.

A. Public Information about CTKD

As described in the Introduction, CTKD was introduced in the Bluetooth standard to improve the *usability* of BT and BLE pairing. Before the introduction of CTKD, devices had to separately pair over BT and BLE. While with CTKD, the devices pair once, either over BT or BLE, derive a pairing key for each transport and establish BT and BLE secure sessions [10, p. 280].

Being a standard-compliant feature CTKD has to be supported by all hardware and software Bluetooth vendors. The list of vendors includes Apple [45], Google [6], Cypress [15], Linux [13], Qualcomm [34], and Intel [23]. Notably, Apple presented it as a core and always-on Bluetooth feature during WWDC 2019.

To use CTKD, a device requires few capabilities. It must be a dual-mode device (i.e., support both BT and BLE), has to support Secure Connections, and implement a Bluetooth version among 4.2, 5.0, 5.1, and 5.2. Examples of devices supporting CTKD are laptops, tablets, smartphones, headsets, speakers, and high-end wearable devices. The number of those devices is steadily growing as dual-mode devices are replacing single-mode ones [12].

CTKD employs the same deterministic *key derivation function (KDF)* for BT and BLE [10, p. 1658]. The KDF takes as inputs a 128-bit (16-byte) key and two 4-byte strings and derives a 128-bit (16-byte) key. If CTKD is started from BLE, then the BT pairing key is derived using the "tmp2" and "brle" strings. In the other case, the derivation is performed using the "tmp1" and "lebr" strings. The key derivation function is deterministic, as using CTKD on the same input key will always generate the same output key. We re-implemented KDF to validate our analysis, see Section V-C for more details.

The Bluetooth standard lacks a security analysis of CTKD but provides only a *limited* and *version-specific* security argument. Since version 5.1 the standard states that "While performing cross-transport key derivation, if the key for the other transport already exists, then the devices shall not overwrite that existing key with a key that is weaker in either strength or MITM protection" [10, p. 1401]. In other words, an attacker cannot overwrite a pairing key with CTKD if the overwriting key has either a lower entropy (i.e., strength) or a lower MitM protection. While this can be expected to protect against attacks in limited settings, other scenarios and attacks are still possible. For example, an adversary can still overwrite keys with *equal* strength and MitM protection *without* violating the standard (as we experimentally demonstrate in Section VI-B). In addition to the limited scope of the countermeasure, it is unclear why it was introduced only for 5.1 and 5.2 devices and not for all Bluetooth versions supporting CTKD, and how it should be interpreted when one of the devices does not support Bluetooth 5.1 or 5.2.

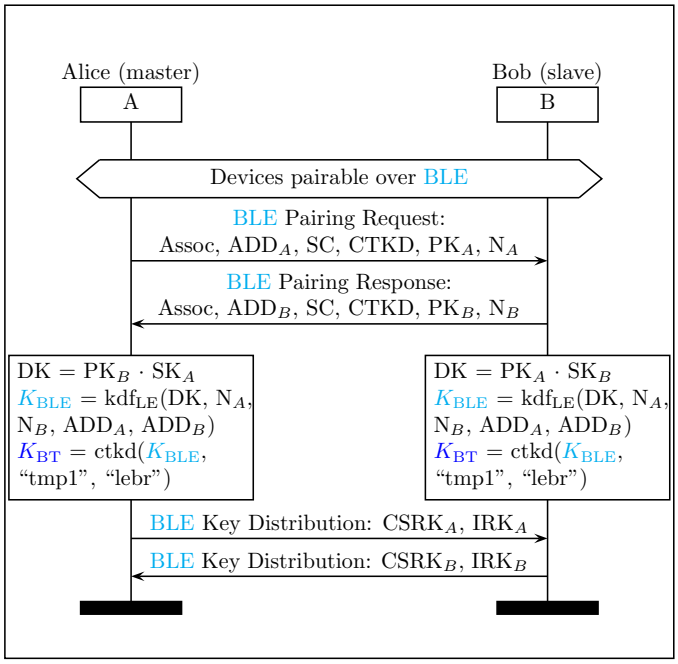

Fig. 1: CTKD from BLE. Alice and Bob negotiate SC and CTKD support during BLE pairing. Then, they compute the BLE pairing key and from that key, they derive the BT pairing key via CTKD (without exchanging any message over BT). Finally, they generate and exchange additional keys for BLE including signature (CSRK) and identity resolving (IRK) keys. After the protocol is completed Alice and Bob can establish secure sessions both for BT and BLE (without having to pair over BT).

B. Reverse-Engineered CTKD Protocols

The public information that we gathered about CTKD, including the ones provided by the Bluetooth standard are *not* sufficient to perform a security analysis of CTKD. Specifically, from the standard is not clear how CTKD is *negotiated* for BT and BLE and if the protocols differ. To address this problem, we reverse-engineered the CTKD negotiation protocols for BT and BLE. Here we present them abstracting the description at the message level. We refer to the Bluetooth master as Alice and to the slave as Bob and in the figure we color-code BLE with light blue and BT with blue. At the end of the section, we detail our RE methodology.

CTKD from BLE Figure 1 shows how CTKD is negotiated and used from BLE to derive BLE and BT pairing keys. Alice and Bob are pairable over BLE and BT and discover each other using BLE scanning and advertising. Then, they perform pairing over BLE using the SMP protocol. We found that CTKD is negotiated by setting to one the Link Key flag of the Initiator and Responder key distribution SMP fields [10, p. 1680] and that such negotiation is not protected. Other than the Link Key

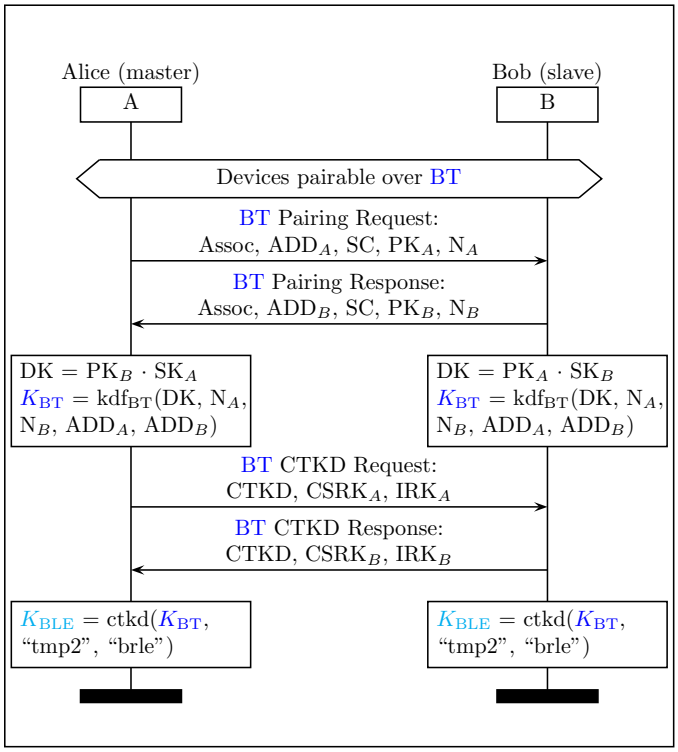

Fig. 2: CTKD from BT. Alice and Bob during BT pairing negotiate SC support. Then, they compute the BT pairing key, start a secure session over BT and send BT CTKD messages containing CTKD support and other keying material generated for BLE such as signature (CSRK) and identity resolving (IRK) keys. Notably, the CTKD request and response are encoded as BLE pairing request and response and tunneled over BT. Afterward, Alice and Bob derive the BLE pairing key, via CTKD (without exchanging any message over BLE). After the protocol is completed Alice and Bob can establish secure sessions both for BT and BLE (without having to pair over BLE).

flag the devices should also declare Secure Connections support (SC) which is also spoofable. The BLE pairing messages also contain an association method (Assoc), a source BLE address (ADD), a public key (PK), and a nonce (N).

After exchanging the pairing messages, the devices compute a Diffie-Hellman shared secret (DK) using the exchanged PK. DK is used to compute the BLE pairing key (K_{BLE}) using BLE pairing key derivation function (kdf$_{LE}$). Then, the devices use CTKD's key derivation function (ctkd) to derive the BT pairing key (K_{BT}). To complete BLE pairing, Alice and Bob establish a secure session over BLE and exchange additional keys (e.g., CSRK and IRK). As a result, Alice and Bob share K_{BLE} and K_{BT}, but they only paired over BLE.

CTKD from BT Figure 2 presents how CTKD is negotiated and used from BT to derive BT and BLE pairing keys. Alice and Bob are pairable over BT and BLE and discover each other via BT inquiry. Then, they exchange pairing request and response messages over BT to negotiate several BT capabilities (including SC), and to exchange their BT addresses, keys, and nonces. Then, they compute DK and use it together with their BT addresses and nonces to compute the BT pairing key (K_{BT}) through the BT pairing key derivation function (kdf$_{BT}$).

Unlike for BLE, BT pairing messages do not include a CTKD flag. What happens is that the devices start a secure BT session and exchange two messages containing the CTKD flag and additional security material needed for BLE such as signature keys (CSRK) and identity resolving keys (IRK). These two messages are peculiar as they are encoded as BLE SMP packets but sent over BT. We are not sure why the Bluetooth standard is not describing such "BLE tunneling" protocol to negotiate CTKD from BT. Once CTKD is negotiated, Alice and Bob use it to derive the BLE pairing key (K_{BLE}) from the BT key and the static strings "tmp2" and "brle".

RE methodology To RE the negotiation and usage of CTKD we used a Linux laptop connected to a dual-mode development board as a test device. The laptop runs a patched Linux kernel capable of pairing diagnostic messages from the board. The board acts as the laptop fronted (i.e., the laptop is the BT/BLE Host while the board is the BT/BLE Controller), and is initialized to report to the laptop all sent and received link-layer traffic using HCI diagnostic messages.

To test CTKD from BLE we sent a BLE pairing request from our test device to a pair of dual-mode headphones (Sony WH-1000XM3) and we monitored the HCI log. To check out CTKD from BT we sent a BT pairing request from our test device to an Android smartphone (Pixel 2) and we monitored the HCI log. In each case, we tested that it was possible to establish BT and BLE secure sessions after only pairing on one transport. Notably, CTKD from BT was particularly tricky to reverse as the CTKD negotiation messages over BT are decoded by Wireshark but appear as standard L2CAP messages.

C. CTKD Cross-Transport Issues (CTI)

We isolated four *cross-transport issues (CTI)* with the specification of CTKD resulting from CTKD bridging BT and BLE without properly enforcing the security boundary between the two. We now describe in detail each CTI.

CTI 1: extended pairing CTKD introduces more options to pair two devices as dual-mode devices are pairable over BT and BLE all the time. This enables an attacker to (silently) pair over a transport that is currently unused. The attacker does not need to wait until a victim is in discoverable mode, as, despite common belief, a Bluetooth device in pairable state already accepts pairing requests.

CTI 2: role asymmetry While BT and BLE roles are defined differently, CTKD does not enforce which role was used to pair on which transport. BT roles can be switched even before pairing, while BLE roles are fixed. This is problematic because an attacker can adversarially switch BT role before using CTKD and send a BT pairing request to a victim which expects BT and BLE pairing responses. We note that, issues with role asymmetry have been already proven effective to bypass BT authentication during session establishment [4].

CTI 3: key tampering CTKD enables to tamper with all BT security keys from BLE and vice versa using only a single run of the pairing protocol. This is a new and powerful attack primitive for Bluetooth. For example, an attacker can use CTKD to write new pairing keys for BT and BLE or

even overwrite trusted pairing keys with her own. Furthermore, by using CTKD from BT the attacker can get access to all BLE security keys distributed as part of BLE pairing including identity resolving key usable to de-anonymize a BLE device.

CTI 4: association manipulation CTKD does not keep track of which association mechanism was used as part of pairing and the negotiation of the association scheme is not protected. Indeed, an attacker can use CTKD to re-establish pairing keys using an arbitrary association scheme. This includes a weak association to write or substitute authenticated keys with unauthenticated ones (e.g., by re-pairing using Just Works). Recently, association confusion attacks have been proposed for BT or BLE [41], CTKD extends this issue across transports.

IV. Attacks via CTKD

We now present our threat model and the design of four novel and standard-compliant attacks on CTKD. Our attacks are the first samples of *cross-transport* exploitation for Bluetooth, as they are capable of exploiting BT and BLE just by targeting either of the two. Moreover, they are the first attacks exploiting CTKD. The attacks do *not* require a strong attacker model. For example, they can be conducted at any time against arbitrary devices (including the ones supporting BT and BLE SC, and SSP with strong association). As our attacks are blurring the security boundary between BT and BLE, we name them *BLUR attacks*.

A. System Model

Our system model considers two victims, Alice and Bob, who can securely communicate over BT and BLE. The victims support CTKD, and are using the most secure BT and BLE modes, namely, SC and SSP with strong association. This setup *should* protect the victims against eavesdropping, impersonation, and man-in-the-middle attacks as claimed in [10, p. 269]. Without loss of generality, we assume that Alice is the master and Bob is the slave.

Regarding the notation, we indicate a BT pairing key with $\mathrm{K_{BT}}$, a BT session key with $\mathrm{SK_{BT}}$, a BLE pairing key with $\mathrm{K_{BLE}}$, a BLE session key with $\mathrm{SK_{BLE}}$. We indicate a Bluetooth address with ADD, a public key with PK, a private key with SK, a shared Diffie-Hellman secret with DK, a nonce with N, and a message authentication code with MAC.

B. Attacker Model and Goals

Our attacker model considers Charlie, an attacker in Bluetooth range with the victims. The attacker's knowledge is limited to what the victims advertise over the air, e.g., full or partial Bluetooth addresses, Bluetooth names, and security and IO capabilities. She can scan and discover devices, send pairing requests and responses, use CTKD, propose weak association mechanisms (e.g., Just Works), and dissect and craft Bluetooth packets. However, the attacker does not know any pairing or session key shared between the victims, and does not have to be present when the victims pair or negotiate a secure session. Moreover, she cannot access and tamper with the victim devices.

The attacker has four goals. (i) *impersonate Alice (master)* and take over her secure sessions with Bob. (ii) *impersonate Bob (slave)* and take over his secure sessions with Alice. (iii) *man-in-the-middle* Alice and Bob' secure session (iv) establish *unintended and stealthy sessions* with Alice and Bob.

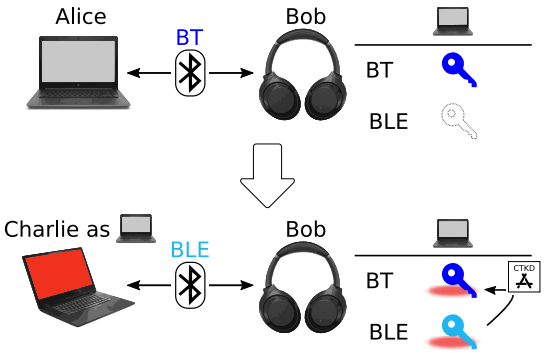

Fig. 3: Attack strategy. Alice and Bob are paired over BT and run a secure BT session. Charlie pairs with Bob as Alice over BLE declaring CTKD support. Then Charlie agrees upon a BLE pairing key with Bob, and, via CTKD, tricks Bob into overwriting Alice's BT pairing key. As a result, Charlie can establish BT and BLE sessions with Bob as Alice, and takes over the real Alice who can no longer connect to Bob. Using a similar strategy, Charlie can also impersonate Bob to Alice, man-in-the-middle Alice and Bob, and establish unintended BT and BLE sessions as an arbitrary device.

Let us clarify some aspects of the attackers' goals. By "take over" we mean that after the attack the bond between the victims is broken (e.g., when Charlie takes over a session from Alice then Alice will not be able to connect back to Bob). Master and slave impersonation are indicated as different goals, as they require different attack strategies. We define an unintended session as a session established with a victim device as an arbitrary device without breaking existing security bonds (e.g., Charlie silently pairs with Alice as a random device using CTKD and connects with Alice over BT and BLE).

C. Attack Strategy

We now describe our attack strategy with the help of Figure 3. Let us assume that Alice is a laptop and Bob is a pair of headphones and the victims are already paired and they are running a secure BT session. Since the victims support CTKD, they are also pairable over BLE, even if the transport is not currently in use. Charlie sends a BLE pairing request to Bob (the victim) pretending to be Alice, and claiming CTKD support. The attacker also declares *no input/output capabilities* to negotiated unauthenticated Just Works (JW) association. This last step does *not* trigger the key overwrite countermeasure described in Section III-A as the attacker is neither changing the MitM protection flag nor the strength (i.e., entropy) of the pairing key.

Bob, even if running a BT session with Alice, has to answer to Charlie with a BLE pairing response as Charlie's message is compliant with the Bluetooth standard. Then, Charlie (as Alice) and Bob agree on a BLE pairing key and, via CTKD, generate a new BT pairing key that *overwrites* Alice's key in Bob's BT key store. In doing so, Charlie, wins two prizes with one shot, as he takes over Alice's BT and BLE sessions with

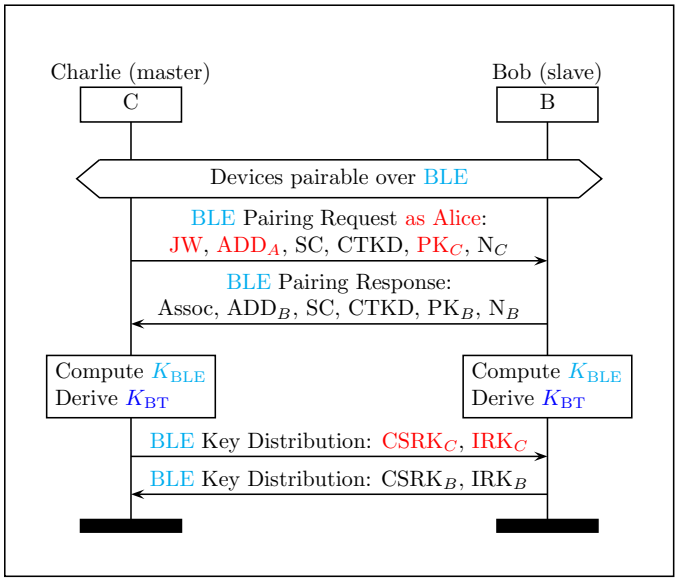

Fig. 4: BLUR master impersonation attack. Charlie sends a BLE pairing request with Alice's address (ADD$_A$) including Just Works (JW) association, CTKD, and his public key (PK$_C$). Bob answers with a BLE pairing response thinking that he is talking to Alice. The attacker and the victim agree on K$_{\text{BLE}}$, and derive K$_{\text{BT}}$, via CTKD and complete BLE pairing by generating and distributing more keys over a secure BLE session. As a result of the master impersonation attack, Charlie tricks Bob into overwriting Alice's keys with his ones and takes over Alice who can no longer connect back to Bob.

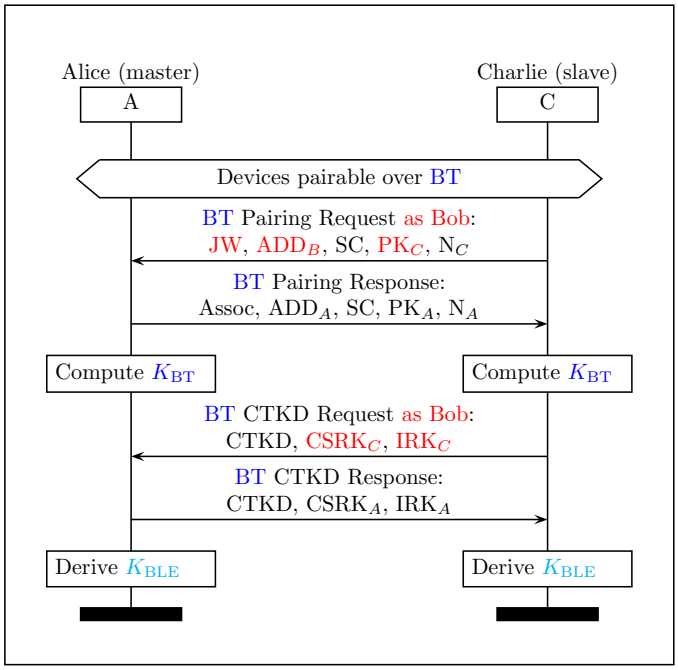

Fig. 5: BLUR slave impersonation attack. Charlie sends a BT pairing request with Bob's address (ADD$_B$) including Just Works (JW) association, and his public key (PK$_C$). The pairing request is valid as BT enables to dynamically switch from slave to master before sending a pairing request. Alice answers with a BT pairing response believing that she is talking to Bob. The attacker and the victim establish K$_{\text{BT}}$, negotiate CTKD and exchange additional keying material for BLE with a BT CTKD request and response messages, and derive K$_{\text{BLE}}$. As a result of the slave impersonation attack, Charlie tricks Alice into overwriting Bob's keys with his ones and takes over Bob who can no longer connect back to Alice.

Bob. In other words, Alice can no longer connect to Bob as she does not know the BT and BLE pairing keys (overwritten by the attacker). Furthermore, Charlie also overwrites other security keys that are distributed during pairing, including CSRK (signature key) and IRK (MAC randomization key). We note that the overwrite trick is transparent to the end user as the standard does not mandate to notify the user about CTKD, and works even if Alice and Bob are sharing BT *and* BLE pairing keys before the attack takes place.

Following a similar strategy, Charlie can impersonate Bob to Alice, man-in-the-middle them, and create unintended sessions as an arbitrary device with a victim. We note that our attack strategy is effective because the Bluetooth standard does not enforce important security properties at the boundary between BT and BLE and does not address all cross-transport threats in its threat model (see Section III-C for more details). In the remaining of this section, we describe the technical details of the four BLUR attacks.

D. Impersonation Attacks

Master impersonation Charlie impersonates Alice and takes over her BT and BLE sessions with Bob as in Figure 4. Bob is already paired with Alice, and can run a BT session with her while Alice's impersonation takes place. Notably, Bob must be pairable over BT and BLE to support CTKD from BT and BLE. Charlie takes advantage of that and sends a BLE pairing request as Alice by using Alice's Bluetooth address (ADD$_A$), Just Works (JW) association while pairing, his public key (PK$_C$), and CTKD support.

As Charlie's BLE pairing request is standard-compliant, Bob sends back a BLE pairing response believing that Alice wants to pair (or re-pair) over BLE using CTKD. Then, Charlie and Bob compute K$_{\text{BLE}}$, derive K$_{\text{BT}}$ via CTKD, and exchange additional BLE key material (e.g., CSRK, IRK) over a BLE secure session. After the master impersonation attack is completed Charlie takes over Alice's BT and BLE sessions by tricking Bob into overwriting Alice's BT and BLE keys with his ones.

Slave impersonation Charlie impersonates Bob and takes over his BT and BLE sessions with Alice as in Figure 5. Alice and Bob have already paired and can run a BLE secure session while the impersonation takes place. Alice has to be pairable over BT and BLE to provide CTKD support from both transports, and Charlie takes advantage of that by sending a BT pairing request to Alice as Bob using Bob's address (ADD$_B$), Just Works (JW), and his public key (PK$_C$). Charlie's pairing request is still standard-compliant even if Charlie is supposed to be the slave as BT, unlike BLE, enables a slave to switch to a master role before sending a pairing request.

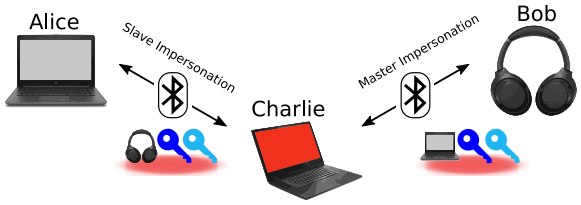

Fig. 6: BLUR MitM attack. Charlie combines the master and slave impersonation attacks presented so far to establish a man-in-the-middle position between Alice and Bob both on BT and BLE.

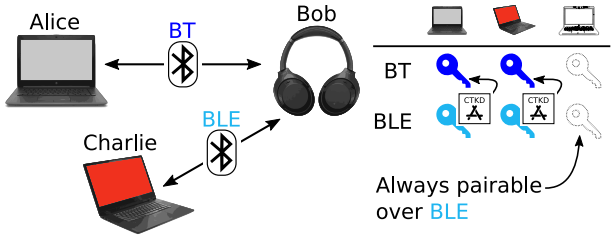

Fig. 7: BLUR unintended sessions attack. Charlie can take advantage of CTKD to establish unintended BT and BLE session with Bob as a random device with arbitrary capabilities. The same can happen if Charlie targets Alice.

Alice answers with a BT pairing response believing that Bob wants to re-pair over BT, and the two agree on K_{BT}. Then, Charlie starts a secure BT session and sends a tunneled BLE pairing request to Alice still pretending to be Bob. The BLE pairing request includes CTKD support and Charlie's signature and MAC randomization BLE keys ($CSRK_C$, IRK_C). Alice answers with a BLE pairing response tunneled over BT and the two derives K_{BLE} via CTKD. Once the slave impersonation attack is completed, Charlie takes over Bob's BT and BLE sessions by tricking Alice into overwriting Bob's BT and BLE keys with his ones.

Man-in-the-middle Charlie can conveniently combine the described master and slave attacks to launch a cross-transport man-in-the-middle attack as shown in Figure 6. If Alice and Bob are running a BLE session, Charlie starts with the slave impersonation attack presenting to Alice as Bob over BT. Otherwise, he launches a master impersonation attack by targeting Bob as Alice over BLE. After the first impersonation attack, the impersonated victim is taken over and disconnects from the other victim. Then, Charlie targets the impersonated victim with a second impersonation attack and establishes a MitM position between the two victims. As a result, Charlie controls all BT and BLE secure sessions between Alice and Bob.

E. Unintended Session Attacks

CTKD enables unintended session attacks as it provides *more ways* to (silently) pair devices. In particular, two devices supporting CTKD are always pairable over BT and BLE, but typically they use one transport at a time. Hence, the attacker can target the unused transport as a stepping stone to establish trusted bonds on BT and BLE via CTKD, while impersonating a random device. To the best of our knowledge, this attack technique is new in the context of Bluetooth.

Let us see how an unintended session attack works in a scenario where Alice and Bob are already paired and are running a secure BT session (see Figure 7). Charlie targets Bob by sending a BLE pairing request using a random Bluetooth address, CTKD support, and Just Works for association. Bob answers to Charlie's request and the two negotiate K_{BLE}, and derive K_{BT} via CTKD. Now, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking Bob's existing sessions (e.g., with Alice) and by using an anonymous identity and arbitrary capabilities. Using a similar strategy, Charlie can reach the same goals targeting Alice.

An unintended session attack is valuable for various reasons. It is *stealthy* as the attacker can establish a trusted relation with the victim as an anonymous device and with minimal user interaction (e.g., Just Works). Moreover, it allows *complete device enumeration* as the attacker, being a trusted peer, can access all BT and BLE services, including the protected ones unlike other attacks such as [14]. Additionally, the attack deterministically *de-anonymizes* BLE devices, as the attacker get access to the identity resolving key distributed during pairing. Finally, it enables the attacker to reach *more* Bluetooth stack code sections than an untrusted device, including remote code execution bugs in the pairing and secure session code.

F. Attacks Discussion

We now discuss the attacks' root causes, their effectiveness regardless of the Bluetooth version, association method, and even CTKD support. Finally, we detail how we discovered them.

Root causes (CTIs) The BLUR attacks take advantage of the four *CTI* presented in Section III-C. As shown in Table I, a checkmark (✓) indicates that a CTI is required, an "*x*" if it is not needed, and an "*" if it is sometimes needed. From the table we see that all attacks exploit extended pairability (CTI 1). The slave impersonation and MitM attacks take advantage of role asymmetries (CTI 2), while some unintended session attacks take advantage of that. Key tampering (CTI 3) is exploited in all attacks as the attacker has to either write or overwrite keys using CTKD. Association manipulation (CTI 4) is required in the first three attacks when the victim expects a strong association mechanism but the attacker negotiates Just Works.

	CTI 1	CTI 2	CTI 3	CTI 4
Master Impersonation	✓	x	✓	*
Slave Impersonation	✓	✓	✓	*
MitM	✓	✓	✓	*
Unintended Session	✓	*	✓	x

TABLE I: Mapping the BLUR attacks to the CTI from Section III-C. CTI 1: extended pairing, CTI 2: role asymmetry, CTI 3: key tampering, and CTI 4: association manipulation. We use a ✓ if a CTI is required to conduct an attack, a x if is not required and a * if is only required in specific cases.

Bluetooth v5.1/5.2 In Section III-A we describe a version-specific key overwrite security argument included in the Bluetooth standard since v5.1. The Bluetooth SIG currently uses this argument to state that the BLUR attacks are *not* effective against Bluetooth 5.1 and 5.2 devices (see https://tinyurl.com/vxhwftc2). We *disagree* with this statement, and we provided them empirical evidence (see Section VI-B) and solid arguments (described below) about why this is not the case.

In particular, the key overwrite countermeasure is ineffective as it is *out of scope* with our attacks. Firstly, it does not cover *non key overwrite* cross-transport attacks such as the cross-transport unintended session and key writing attacks presented in this work. In addition, it does not protect against key overwrite attacks *not* involving the downgrade of keys' strength and MitM protection. Specifically, our key overwrite attacks declare "no input/output capabilities" to force the usage of Just Works without downgrading key strengths or MitM protections.

Association The BLUR attacks are effective regardless of the association methods supported by a victim, as the attacker can *always* downgrade it to Just Works. Even Just Works might require minimal user interaction (e.g., Yes/No pairing prompt) but remains an *unauthenticated* and vulnerable mechanism (e.g., the user has no way to tell if the remote pairing device is trustworthy).

CTKD support Interestingly our attacks can be launched even if one of the victims does *not* support CTKD. By design CTKD's negotiation is not protected and enforced across pairing sessions. Hence, if the attacker impersonates a device not supporting CTKD, she can still use the BLUR attacks if the victim supports CTKD. For example, the adversary can impersonate BT-only speakers to a BT/BLE (dual-mode) laptop and exploit CTKD.

Discovery We discovered the BLUR attacks by *inference* from the public information and RE details presented in Section III. Our experiments involved actual devices and static and dynamic analysis of the exchanged Bluetooth packets and the CTKD code.

V. Implementation

In this section we describe our attack scenario, our implementation of a custom attack device to perform the BLUR attacks and our re-implementation of CTKD's key derivation function. We will open-source both implementations.

A. Attack Scenario

Our attack scenario follows the example in Figure 8 and includes two victims, Alice (master) and Bob (slave). Alice is represented by a 7th generation ThinkPad X1 laptop and Bob by a pair of Sony WH-CH700N headphones. The attacker (Charlie) uses a CYW920819 development board [16] and a 3rd generation ThinkPad X1 laptop as an attack device. The implementation of the attack device is presented in Section V-B. In our evaluation, presented in Section VI, we use the same attack scenario to attack other victim devices.

Table II summarizes the most relevant features of Alice, Bob, and Charlie. Alice and Bob have an Intel Bluetooth chip, while Bob has a Cambridge Silicon Radio (CSR) one. Alice, Bob, and Charlie support respectively Bluetooth 5.1, 4.1, and 5.0. Alice and Charlie support Secure Connections both on the Host and the Controller, while Bob only on the Controller. All devices support BT, BLE, and CTKD. Regarding pairing association methods, the laptops support Numeric Comparison, while the headsets only support Just Works as they lack a display.

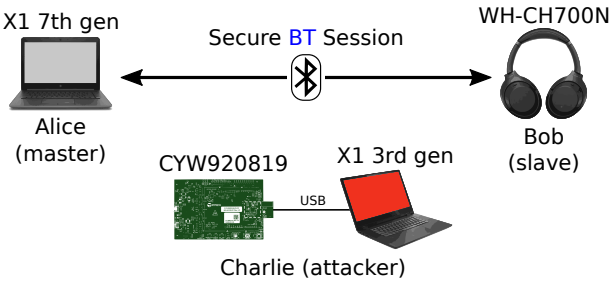

Fig. 8: BLUR attack scenario. Alice (master) is a ThinkPad X1 7th gen, Bob (slave) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in absence of Charlie, and are running a secure BT session.

	Alice	Bob	Charlie
Device(s)	X1 7th gen	WH-CH700N	X1 3rd gen / CYW920819
Radio Chip	Intel	CSR	Intel / Cypress
Subversion	256	12942	256 / 8716
Version	5.1	4.1	5.0
Name	x7	WH-CH700N	x1
ADD	Redacted	Redacted	Redacted
Class	0x1c010c	0x0	0x0
BT SC	True	Only Controller	True
BT AuthReq	0x03	0x02	0x03
BLE SC	True	True	True
BLE AuthReq	0x2d	0x09	0x2d
CTKD	True	True	True
h7	True	False	True
Role	Master	Slave	Master
IO	Display	No IO	Display
Association	Numeric C.	Just Works	Numeric C.
Pairable	True	True	True

TABLE II: Relevant features of Alice, Bob, and Charlie. We redact the devices' Bluetooth addresses for privacy reasons.

B. Attack Device

To conduct our attacks we developed an attack device making use of a *CYW920819 development board* connected to a *Linux laptop* (see Figure 9). The devices support BT, BLE, SC, and CTKD. We picked these devices as COTS devices do not allow to modify their Bluetooth firmware (Controller) but at most the OS Bluetooth stack (Host). A software-defined radio (SDR) is also out of scope because there is no open-source BT/BLE SDR stack currently available.

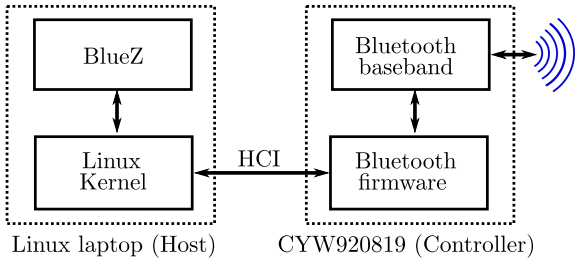

Fig. 9: Attack device block diagram. The attack device is composed of Linux laptop (Host) and a CYW920819 development board (Controller) connected via USB and communicating using the Host Controller Interface (HCI) protocol.

Instead, with our attack device, we can program our development board (Bluetooth Controller) to impersonate any BT/BLE device, we can patch its closed-source firmware to control both BT LMP and BLE LL link layer packets. Moreover, we can alter the laptop's BT and BLE kernel and user-space code to set Bluetooth Host-specific configuration bits such as negotiating CKTD and Just Works. We now describe in detail how we modify the attack device's Host and Controller components.

Host modifications For the host, we use standard Linux tools to configure an Bluetooth interface (e.g., `hciconfig`), and to discover and pair with a device (e.g., `bluetoothctl`, `hcitool` and `btmgmt`). In particular, `btmgmt` was very useful as it provides handy low-level commands. For example, it includes commands to toggle BT, BLE, SC, scanning, and advertising. Moreover, it allows to easily send custom pairing requests on BT and BLE and to set the related association (e.g., Just Works).

Furthermore, we configured our host to get all link-layer packets sent and received by the controller. This is handy as it enables to monitor both HCI and link-layer packets directly from the host (e.g., using Wireshark). To activate link-layer packet forwarding, we sent a proprietary Cypress HCI command from the host to the controller that switches on an undocumented diagnostic mode in the controller. Then, we added extra C code to the Linux kernel to parse those special HCI packets in the host.

Controller modifications We modified the controller by dynamically patching the development board Bluetooth firmware using a Cypress proprietary mechanisms. To patch the firmware we had to extract it from the board and statically reverse-engineer its relevant parts. In particular, to extract the firmware we used a proprietary HCI command to read and save a runtime RAM snapshot from the board's SoC. We use the memory maps that we extracted from the board's SDK to extract the memory segments from the snapshot (e.g., ROM, RAM, and the scratchpad). As expected, the firmware was in the ROM segment and was a stripped ARM binary containing 16-bit Thumb instructions.

To reverse-engineer the firmware, we loaded the ROM, RAM, and scratchpad in Ghidra and statically analyzed them. In our first pass, we isolated the libc functions (e.g., `malloc` and `calloc`) by looking at the signatures and the code patterns of the functions that are called the most. Then, we found the firmware debugging symbols hidden in the board's SDK and loaded them into Ghidra. Using these symbols we isolated functions and data structures relevant to the BLUR attacks. Then, we wrote ARM Thumb assembly patches to change their behaviors and we apply those patches at runtime using internalblue [30], an open-source toolkit to manage several Bluetooth devices including our board. Our set of patches allows transforming our board in whatever device we want by changing its identifiers including addresses, names, and capabilities,

C. CTKD Key Derivation Function (KDF)

We implemented CTKD's custom KDF, following the Bluetooth standard [10, p. 1401]. This implementation is *not* required to conduct the attack, but it was used to check that the CTKD keys were correctly derived. Our implementation is written in Python 3, uses the PyCA cryptographic module [7], and was successfully tested against the test vectors in the standard [10, p. 1721]. We now describe its technical details.

The KDF computes $\mathrm{K_{BLE}}$ using $\mathrm{K_{BT}}$, "tmp2" and "brle", and $\mathrm{K_{BT}}$ from $\mathrm{K_{BLE}}$, "tmp1" and "lebr". Each case can be represented by a system of equations (see below). If CTKD is run from BT then the first system of equations is used otherwise the second. Each system has two equations and the top equation is used if both devices support the *h7* key conversion function. h7 is negotiated during pairing using the AuthReq flag [10, p. 1634]. All four equations internally use $f(a, b)$ that is implemented as AES-CMAC$(key, plaintext)$.

$$K_{BLE} = \begin{cases} f\left(f\left(tmp2, K_{BT}\right), brle\right) & \text{if h7 is supported} \\ f\left(f\left(K_{BT}, tmp2\right), brle\right) & \text{otherwise} \end{cases}$$

$$K_{BT} = \begin{cases} f\left(f\left(tmp1, K_{BLE}\right), lebr\right) & \text{if h7 is supported} \\ f\left(f\left(K_{BLE}, tmp1\right), lebr\right) & \text{otherwise} \end{cases}$$

VI. Evaluation

In this section we present how we successfully conducted the BLUR attacks on 16 devices using 14 unique Bluetooth chips. Our results confirm that the BLUR attacks are effective against different device types (e.g., laptops, smartphones, headphones, and development boards), manufacturers (e.g., Samsung, Dell, Google, Lenovo, and Sony), operating systems (e.g., Android, Windows, Linux, and proprietary OSes), and Bluetooth firmware (e.g., Broadcom, CSR, Cypress, Intel, Qualcomm, and Samsung).

A. Setup

The BLUR attacks, presented in Section IV, include master impersonation, slave impersonation, man-in-the-middle, and unintended session attacks. In the next paragraphs, we describe how we conducted each attack using the attack device described in Section V-B.

Laptop (master) BLUR impersonation attack To impersonate the laptop, we patch our attack device to clone the laptop's Bluetooth features (e.g., Bluetooth address, name, device class, and security parameters) Then, we send a BLE

Device			Chip		Bluetooth	BLUR Attack			
Producer	Model	OS	Producer	Model	Version	Role	MI/SI	MitM	US
Cypress	CYW920819EVB-02	Proprietary	Cypress	CYW20819	5.0	Slave	✓	✓	✓
Dell	Latitude 7390	Win 10 PRO	Intel	8265	4.2	Slave	✓	✓	✓
Google	Pixel 2	Android	Qualcomm	SDM835	5.0	Slave	✓	✓	✓
Google	Pixel 4	Android	Qualcomm	702	5.0	Slave	✓	✓	✓
Lenovo	X1 (3rd gen)	Linux	Intel	7265	4.2	Slave	✓	✓	✓
Lenovo	X1 (7th gen)	Linux	Intel	9560	5.1	Slave	✓	✓	✓
Samsung	Galaxy A40	Android	Samsung	Exynos 7904	5.0	Slave	✓	✓	✓
Samsung	Galaxy A51	Android	Samsung	Exynos 9611	5.0	Slave	✓	✓	✓
Samsung	Galaxy A90	Android	Qualcomm	SDM855	5.0	Slave	✓	✓	✓
Samsung	Galaxy S10	Android	Broadcom	BCM4375	5.0	Slave	✓	✓	✓
Samsung	Galaxy S10e	Android	Broadcom	BCM4375	5.0	Slave	✓	✓	✓
Samsung	Galaxy S20	Android	Broadcom	BCM4375	5.0	Slave	✓	✓	✓
Xiaomi	Mi 10T Lite	Android	Qualcomm	9312	5.1	Slave	✓	✓	✓
Xiaomi	Mi 11	Android	Qualcomm	10765	5.2	Slave	✓	✓	✓
Sony	WH-1000XM3	Proprietary	CSR	12414	4.2	Master	✓	✓	✓
Sony	WH-CH700N	Proprietary	CSR	12942	4.1†	Master	✓	✓	✓

† CTKD was backported by the vendor to Bluetooth 4.1.

TABLE III: BLUR attacks evaluation results. The first three columns show the device's producer, model, and OS. The next two columns state the Bluetooth chip's producer and model. The sixth column tells the Bluetooth version of the target device. The seventh column indicates the attacker role. The last three columns contain a checkmark (✓) if a device is vulnerable to the relevant BLUR attack.

pairing request from the attack device to the headphones declaring CTKD and Just Works support. The malicious BLE pairing request is sent using `btmgmt`'s text-based user interface (TUI). The headphones accept the pairing request, and the devices agree on $\mathrm{K_{BLE}}$, derive $\mathrm{K_{BT}}$ via CTKD and establish a secure BLE session. Then, the headphones terminate the BT session with the impersonated laptop and establish a secure BT session with the attack device. The impersonated laptop cannot connect back with the headphones as it does not possess the correct pairing keys overwritten by the attacker.

Headphones (slave) BLUR impersonation attack To impersonate the headphones, we patch our attack device to clone the headphones' Bluetooth features. Then, we send a BT pairing request from the attack device to the laptop declaring CTKD and Just Works support using `btmgmt`'s TUI. The laptop accepts to pair over BT as a BLE slave can send a BT pairing request as a master. The devices agree on $\mathrm{K_{BT}}$, derive $\mathrm{K_{BLE}}$via CTKD, and establish a secure session over BT. The impersonated headphones cannot connect to the laptop as they do not own the correct pairing keys.

BLUR Man-in-the-middle attack By using two development boards connected to the same laptop, we can impersonate the laptop and the headphones at the same time, and man-in-the-middle them. In particular, we run the laptop (master) impersonation attack first, and then the headphone (slave) impersonation attack. As a result, the attack device positions itself in the middle between the victims.

BLUR Unintended sessions attack For the unintended session attack, we patched our attack device to look like an unknown device to the current victim (e.g., unknown Bluetooth address and name). If the victim is a master, we run the same steps used in the slave impersonation attack otherwise we use the master impersonation attack's steps. In both cases, the attacker completes pairing using CTKD and can establish secure sessions over BT and BLE with the victim.

B. Results

We exploited the BLUR attacks against 16 unique devices employing 14 different Bluetooth chips and covering all Bluetooth versions supported by CTKD. We show our results in Table III. The table's first six columns indicate the device's producer, model name, operating system, chip manufacturer, chip model, and Bluetooth version. The seventh column contains either Slave if the attacker's role is slave, or Master otherwise. The table's last three columns have a checkmark (✓) if a device is vulnerable to master or slave impersonation attack (MI/SI), MitM, or unintended session (US) attack.

Table III empirically demonstrates that the BLUR attacks are *effective* on actual devices and are *compliant* with the Bluetooth standard. The attacks are effective on all Bluetooth versions supporting CTKD (i.e., Bluetooth 4.2, 5.0, 5.1, and 5.2). In addition, they succeed regardless of the hardware and software details of the victim device. Interestingly, they work even on older Bluetooth versions to which CTKD was backported.

Moreover, Table III experimentally confirms that Bluetooth 5.1/5.2 are *still vulnerable* to the BLUR attacks, despite the CTKD key overwrite mitigation in the standard [10, p. 1401]. See Section III-A for an introduction and Section IV-F for our

explanation about why it is ineffective. We want to evaluate more 5.1/5.2 devices supporting CTKD, such as speakers and headsets. So far, we were able to find only 5.1/5.2 high-end devices supporting CTKD (e.g., Xiaomi Mi 11, Mi 10T Lite, and ThinkPad X1 7th gen).

VII. Discussion

A. Countermeasures

To *concretely* address the BLUR attacks and their root causes (CTIs), we present four countermeasures. As they act at *the protocol level*, they are effective regardless of the Bluetooth version number, and as they also cover non key-downgrade attacks, they block all BLUR attacks. This is not the case with the Bluetooth SIG's 5.1 key overwrite countermeasure described in Section III-A.

C1: Disable pairing when not needed To prevent an attacker from pairing with a device on unused transports, a device should automatically stop being pairable on a transport that is not currently in use. To avoid DoS issues, a device should also allow a user to enable and disable pairing manually on a specific transport.

C2: Align BT and BLE roles To fix role asymmetries when using CTKD from BT or BLE, a device should store the transport and the role used while pairing, and enforce it across re-pairings. In case of a role mismatch, the device should abort pairing. We note that the BIAS paper [4] also takes advantage of role switching but is not proposing role switch enforcement as a countermeasure.

C3: Prevent cross-transport key tampering To prevent cross-transport key overwrites via CTKD, a device should disable it if a pairing key already exists for the other transport. To overwrite a pairing key a user should explicitly re-pair on that transport. To mitigate cross-transport key writes, CTKD should be disabled when two devices, who already share a pairing key on a transport, re-pair on that transport with a weaker key (that would be used as input to CTKD). A key is stronger than another one if its entropy is higher or if is established with a stronger association mechanism.

C4: Enforce strong association mechanisms To prevent an attacker from manipulating associations across transports, a device should keep track of the association mechanism used while pairing for the first time and enforce it for subsequent re-pairings (across BT and BLE). There is no reason why two devices which support strong association would want to ever use a weaker association scheme. If a weaker mechanism than the one stored is proposed, pairing should be aborted.

The four countermeasures ultimately block the BLUR attacks. In particular, C3 prevents impersonation and MitM as the attacker cannot write or overwrite keys across transports, but only target BT and BLE separately. To stop the unintended session attacks, C1 is also needed as the attacker should not be able to pair with CTKD on unused transports. C2 and C4 help mitigate the attacks by providing more defense-in-depth, but they are not strictly required.

Our countermeasures are easy to implement and do not rely on backward-incompatible features. In particular, they can be implemented in the *Bluetooth Host* (i.e., OS level). C2, C3, and C4 can be realized by keeping track of extra data that is exchanged during pairing (e.g., device role, association) and aborting the protocol when needed. Logging is already supported and used by the Host (e.g., to store pairing keys). C1 can be implemented with a timer that disables pairability on a transport when not needed and a simple user interface to monitor and switch on/off pairability for BT and BLE.

PoC for C3 To verify the effectiveness of C3 and show that our mitigations are easy to implement and do not impact normal operations, we developed a proof-of-concept (PoC) for C3 for Linux. We can evaluate multiple device classes simultaneously by testing Linux, as it is employed by Android smartphones, embedded devices, and laptops. The C3 PoC works as follows. We pair a Linux laptop (victim) with an arbitrary device with CTKD. Then, to disable key overwrites on the laptop, we unset the write permission bit of the pairing key file stored at `/var/lib/bluetooth`. We then use the paired devices normally to demonstrate no impact on benign use. Finally, we run the BLUR impersonation attack and we confirm that it is ineffective as the attacker cannot overwrite the pairing keys.

B. Lessons Learned

Specification and modeling Security mechanisms that *cross* the security boundary between two technologies should be well-specified and tested against a comprehensive cross-transport threat model. On the contrary, the Bluetooth standard provides an incomplete specification for CTKD and only discusses some cherry-picked cross-transport threats.

Security guarantees Cross-transport mechanisms should be designed such that the mechanisms trusted at the boundary between the two transport (i.e., BT and BLE pairing) have the *same* threat model and provides *equivalent* security guarantees. This is not the case for Bluetooth as BT and BLE use different pairing protocols, link layer mechanisms, and threat models.

Usability vs. Security CTKD was introduced to improve Bluetooth's usability, but, in light of the presented attacks, the usability benefits are not balancing the security issues deriving from CTKD. Indeed, it is paramount to find a good balance between usability and security and *not* trade off the latter for the former.

VIII. Related Work

Bluetooth provides a royalty-free and widely-available cable replacement technology [20]. Standard-compliant attacks on Bluetooth are particularly dangerous as all devices are affected, regardless of version numbers or implementation details. Such attacks were discovered since Bluetooth v1.0 [24], [29].

The Bluetooth standard evolved over time to include better pairing mechanisms (e.g., SSP, SC) and two transports (e.g., BT, BLE). Recent standard-compliant attacks on BT include attacks on legacy pairing [39], secure simple pairing (SSP) [21], [40], [9], association [22], [41], key negotiation [2], and authentication procedures [28], [42], [4]. Regarding-BLE we have attacks on legacy pairing [36], key negotiation [5], SSP [9], [46], reconnections [44], and GATT [25].

The BLUR attacks are novel compared to *prior* standard-compliant attacks. As we can see from Table IV they are the first *cross-transport* attacks, meaning the first targeting BT

Year	Paper	Target	Attack				Note
			Phase	C I A K	SC/SCO	Persistent	
Attacks on BT							
2016	Albazrqaoe et al. [1]	Standard	Any	◐○○○	x	-	BlueEar Sniffer
2017	Seri et al. [37]	Impl.	Pairing	●●●○	NA	✓	BlueBorne
2018	Sun et al. [40]	Standard	Pairing	●●●○	✓	-	Passkey (MitM)
2018	Biham et al. [9]	Impl.	Pairing	●●●◐	NA	✓	Fixed Coordinate Invalid Curve
2019	Ossmann et al. [32]	Standard	NA	◐○○○	x	-	Ubertooth sniffer
2019	Antonioli et al. [2]	Standard	Pairing	●●◐○	✓	-	KNOB (MitM)
2020	Antonioli et al. [4]	Standard	Pairing	●●●○	✓	-	BIAS
2021	Tschirschnitz et al. [41]	Standard	Pairing	●●●○	✓	-	Method Confusion (MitM)
Attacks on BLE							
2016	Jasek et al. [25]	Standard	NA	◐○○○	x	-	Black Hat
2019	Seri et al. [38]	Impl.	NA	○◐◐○	NA	✓	Bleedingbit
2020	Zhang et al. [46]	Standard	Pairing	◐◐◐○	✓	-	MitM (SCO)
2020	Wu et al. [44]	Standard	Session	○○●○	✓	-	BLESA
2020	Garbelini et al. [19]	Impl.	Any	◐◐◐○	NA	-	SweynTooth fuzzer
2020	Antonioli et al. [5]	Standard	Pairing	●●◐○	✓	-	Downgrade (MitM)
Cross-transport attacks on BT and BLE							
2021	**BLUR (this work)**	Standard	Any	●●●◐	✓	✓	**First against CTKD**

TABLE IV: Comparison with related work. The BLUR attacks are the first *cross-transport* standard-compliant attacks for Bluetooth and the first targeting *CTKD*. C = Data Confidentiality, I = Data Integrity, A = Device Authentication, K = Key disclosure. No (○) Partially (◐), Yes (●).

from BLE and vice versa. Moreover, no prior attacks targeted (and evaluated the security of) CTKD. Finally, like the BIAS attack [4], they require a weak threat model as the attacker can target a victim at any time. Unlike the BIAS attack, the effect of our attacks is persistent across sessions. Like other standard-compliant attacks, the BLUR attacks are effective regardless of the security mode (e.g., SSP with SC), association method (e.g., Numeric Comparison), and Bluetooth version numbers.

We have seen attacks targeting specific implementation flaws on BT [37] and BLE [38], [19]. As our attacks target the specification level, they are effective regardless of the implementation details. Several surveys on BT and BLE security were published [17], [31], [33] but neither of those surveys nor the Bluetooth standard considers CTKD as a threat. We here demonstrate that CTKD is a serious threat and must be included in the standard Bluetooth threat model.

Cross-protocols attacks were exploited for proximity technologies using Bluetooth and Wi-FI. Two prominent examples are attacks on Apple ZeroConf [8] and Google Nearby Connections [3]. However, no prior attack targeted the BT/BLE combination.

The cryptographic primitives used by Bluetooth have been extensively analyzed. For example, the E_0 cipher used by BT was investigated [18] and it is considered relatively weak [33]. SAFER+, used for authentication, was analyzed as well [27]. BT and BLE "Secure Connections" use the AES-CCM authenticated-encryption cipher. AES-CCM was extensively analyzed [26], [35] and it is FIPS-compliant. As our attacks are at the protocol-level, they are effective even with perfectly secure cryptographic primitives.

IX. Conclusion

This work presents the first security evaluation of CTKD. CTKD was introduced in the Bluetooth standard to improve the usability of pairing. With CTKD two devices can pair on BT (or BLE) and generate pairing keys for both transports. CTKD is a novel attack surface as it allows to tamper with BT from BLE and vice versa, and is only partially documented in the Bluetooth standard (without an appropriate security analysis).

To address these issues, we RE the CTKD protocols and analyzed them using a *cross-transport* attacker model. Our analysis uncovers four critical cross transport issues (CTI) in the specification of CTKD. As such, *all* Bluetooth devices supporting CTKD are currently affected by those vulnerabilities.

We leverage the CTIs to implement four standard-compliant cross-transport attacks. Our attacks allow an attacker to impersonate and MitM devices, and allow establishing unintended (anonymous) sessions with a victim to enumerate sensitive data and send malicious packets. The attacks are the first standard-compliant BT and BLE attacks to not require the attacker to be present when a victim is pairing or establishing a secure session, unlike prior work [22], [21], [36], [40], [9], [2], [5], [4], [44], [46], [41]. In particular, our attacks are the first that can be conducted in absence of one of the victims. The attacks are effective *regardless of* the targeted Bluetooth version and security mode (e.g., SSP, SC, on strong association).

To demonstrate the practicality of the BLUR attacks, we presented a low-cost implementation based on readily available hardware and open-source software. We use our implementation to empirically confirm that the BLUR attacks are standard-compliant and effective all targeted devices. In particular, we exploited 16 different devices using 14 unique Bluetooth chips. Our device sample includes all Bluetooth versions supporting CTKD (e.g., 4.2, 5.0, 5.1, and 5.2) and BT and BLE devices supporting SC and strong association.

To fix the presented attacks and their root causes, we propose protocol-level countermeasures, and demonstrate the efficacy of the most important one (disable key overwrites) experimentally.

REFERENCES

[1] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. Practical bluetooth traffic sniffing: Systems and privacy implications. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*, pages 333–345. ACM, 2016.

[2] 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, August 2019.

[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Nearby Threats: Reversing, Analyzing, and Attacking Google's "Nearby Connections" on Android. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2019.

[4] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. BIAS: Bluetooth Impersonation AttackS. In *Proceedings of Symposium on Security and Privacy (S&P)*. IEEE, May 2020.

[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *Transactions on Privacy and Security (TOPS)*, 2020.

[6] AOSP. Fluoride Bluetooth stack. https://chromium.googlesource.com/aosp/platform/system/bt/+/master/README.md, Accessed: 2020-01-27, 2020.

[7] Python Cryptographic Authority. Python cryptography. https://cryptography.io/en/latest/, Accessed: 2019-02-04, 2019.

[8] Xiaolong Bai, Luyi Xing, Nan Zhang, XiaoFeng Wang, Xiaojing Liao, Tongxin Li, and Shi-Min Hu. Staying secure and unprepared: Understanding and mitigating the security risks of apple zeroconf. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 655–674. IEEE, 2016.

[9] Eli Biham and Lior Neumann. Breaking the bluetooth pairing–fixed coordinate invalid curve attack. http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf, 2018.

[10] Bluetooth SIG. Bluetooth Core Specification v5.2. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726, Accessed: 2020-01-27, 2019.

[11] Bluetooth SIG. Bluetooth Markets. https://www.bluetooth.com/markets/, 2019.

[12] Bluetooth SIG. Bluetooth Market Update 2020. https://www.bluetooth.com/bluetooth-resources/2020-bmu/, 2020.

[13] BlueZ. Bluetooth 4.2 features going to the 3.19 kernel release. https://tinyurl.com/q9dzh2h, Accessed: 2020-01-27, 2014.

[14] Guillaume Celosia and Mathieu Cunche. Fingerprinting bluetooth-low-energy devices based on the generic attribute profile. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, pages 24–31, 2019.

[15] Cypress. BLE and Bluetooth. https://www.cypress.com/products/ble-bluetooth, Accessed: 2020-01-27, 2019.

[16] Cypress. CYW920819EVB-02 Evaluation Kit. https://www.cypress.com/documentation/development-kitsboards/cyw920819evb-02-evaluation-kit, Accessed: 2019-11-16, 2019.

[17] John Dunning. Taming the blue beast: A survey of bluetooth based threats. *IEEE Security & Privacy*, 8(2):20–27, 2010.

[18] Scott Fluhrer and Stefan Lucks. Analysis of the E0 encryption system. In *Proceedings of the International Workshop on Selected Areas in Cryptography*, pages 38–48. Springer, 2001.

[19] Garbelini, Matheus and Chattopadhyay, Sudipta and Wang, Chundong. SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. https://asset-group.github.io/disclosures/sweyntooth/sweyntooth.pdf, Accessed: 2020-04-08, 2020.

[20] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf J Joeressen, and Warren Allen. Bluetooth: Vision, goals, and architecture. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(4):38–45, 1998.

[21] Keijo Haataja and Pekka Toivanen. Two practical man-in-the-middle attacks on Bluetooth secure simple pairing and countermeasures. *Transactions on Wireless Communications*, 9(1):384–392, 2010.

[22] Konstantin Hypponen and Keijo MJ Haataja. "nino" man-in-the-middle attack on bluetooth secure simple pairing. In *Proceedings of the International Conference in Central Asia on Internet*, pages 1–5. IEEE, 2007.

[23] Intel. Intel Wireless Solutions. https://www.intel.com/content/www/us/en/products/wireless.html, Accessed: 2020-01-27, 2019.

[24] Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In *Proceedings of the Cryptographers' Track at the RSA Conference*, pages 176–191. Springer, 2001.

[25] Sławomir Jasek. Gattacking bluetooth smart devices. Black Hat USA Conference, 2016.

[26] Jakob Jonsson. On the security of CTR+ CBC-MAC. In *Proceedings of the International Workshop on Selected Areas in Cryptography*, pages 76–93. Springer, 2002.

[27] John Kelsey, Bruce Schneier, and David Wagner. Key schedule weaknesses in SAFER+. In *Proceedings of the Advanced Encryption Standard Candidate Conference*, pages 155–167. NIST, 1999.

[28] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M Ufuk Çağlayan. Relay attacks on Bluetooth authentication and solutions. In *Proceedings International Symposium on Computer and Information Sciences*, pages 278–288. Springer, 2004.

[29] Andrew Y Lindell. Attacks on the pairing protocol of Bluetooth v2.1. *Black Hat USA, Las Vegas, Nevada*, 2008.

[30] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. InternalBlue - Bluetooth binary patching and experimentation framework. In *Proceedings of Conference on Mobile Systems, Applications and Services (MobiSys)*. ACM, June 2019.

[31] Nateq Be-Nazir Ibn Minar and Mohammed Tarique. Bluetooth security threats and solutions: a survey. *International Journal of Distributed and Parallel Systems*, 3(1):127, 2012.

[32] Michael Ossmann. Project Ubertooth. https://github.com/greatscottgadgets/ubertooth, Accessed: 2019-10-21, 2019.

[33] John Padgette. Guide to bluetooth security. *NIST Special Publication*, 800:121, 2017.

[34] Qualcomm. Expand the potential of Bluetooth. https://www.qualcomm.com/products/bluetooth, Accessed: 2020-01-27, 2019.

[35] Phillip Rogaway. Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, 2011.

[36] Mike Ryan. Bluetooth: With low energy comes low security. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, volume 13, pages 4–4. USENIX, 2013.

[37] Ben Seri and Gregory Vishnepolsky. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/, Accessed: 2018-01-26, 2017.

[38] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. BLEEDINGBIT: The hidden attack surface within BLE chips. https://armis.com/bleedingbit/, Accessed: 2019-07-24, 2019.

[39] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN. In *Proceedings of the conference on Mobile systems, applications, and services (MobiSys)*, pages 39–50. ACM, 2005.

[40] Da-Zhi Sun, Yi Mu, and Willy Susilo. Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard v5. 0 and its countermeasure. *Personal and Ubiquitous Computing*, 22(1):55–67, 2018.

[41] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. Method Confusion Attack on Bluetooth Pairing. In *Proceedings of Symposium on Security and Privacy (S&P)*. IEEE, 2021.

[42] Ford-Long Wong, Frank Stajano, and Jolyon Clulow. Repairing the Bluetooth pairing protocol. In *Proceedings of International Workshop on Security Protocols*, pages 31–45. Springer, 2005.

[43] Joshua Wright. I Can Hear You Now - Eavesdropping on Bluetooth Headsets. https://www.willhackforsushi.com/presentations/icanhearyounow-sansns2007.pdf, Accessed: 2018-10-30, 2018.

[44] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *14th USENIX Workshop on Offensive Technologies (WOOT)*, 2020.

[45] Apple WWDC. What's New in Core Bluetooth. https://developer.apple.com/videos/play/wwdc2019/901, Accessed: 2020-01-27, 2019.

[46] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 37–54, 2020.