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

Anonymous Author(s)

ABSTRACT

The Bluetooth standard specifies Bluetooth Classic (BT) and Bluetooth Low Energy (BLE). The two transports have different security architectures and threat models and provide dedicated pairing and session establishment protocols. Traditionally, two devices would have to pair over BT and BLE to use both securely. But in 2014, Bluetooth 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 transport. While CTKD crosses the security boundary between BT and BLE, little information is know about CTKD internals and no prior work analyzed its security implications.

In this work, we present the first complete description of CTKD obtained by merging the scattered information from the Bluetooth standard and results from reverse-engineering experiments. Then, we perform a security evaluation of CTKD and uncover four issues in its specification that can be used to cross the security boundary between BT and BLE. We leverage these issues to design four standard-compliant attacks exploiting CTKD and enabling cross-transport Bluetooth exploitation. The attacks work even if the strongest security mechanism for BT and BLE are in place and 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 root causes of the BLUR attacks and present effective countermeasures to fix them. We disclosed our findings and countermeasures to the Bluetooth SIG in May 2020 and received CVE-2020-15802.

1 INTRODUCTION

Bluetooth is a pervasive wireless technology used by billions of devices including mobile phones, laptops, headphones, cars, speakers, medical, and industrial appliances [11]. Bluetooth 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 and voice calls. While BLE is optimized for connection-less and very-low-power use cases such as fitness tracking and digital contact tracing.

The Bluetooth standard defines different security architectures and threat models for BT [10, p. 947] and BLE [10, p. 1617]. Each transport provides pairing and secure session establishment protocols. Pairing results in the establishment of a pairing key and secure session establishment allows paired devices to create a secure channel through a fresh session key derived from their shared pairing key.

Traditionally, two devices would have to pair over BT *and* BLE to securely use both. In 2014, to address this usability issue, Bluetooth v4.2 introduced *Cross-Transport Key Derivation (CTKD)*. 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]. All major Bluetooth software (e.g., Apple, Linux, Android, and Windows) and hardware providers (e.g., Cypress, Intel, Qualcomm, Broadcom, Apple, Sony, and Bose) support CTKD.

Security-wise, CTKD has not received any attention from the research community and the Bluetooth standard hastily describes only some aspects and threats associated with CTKD. On the other hand, CTKD is a very interesting attack surface, as it is a standard-compliant security feature, is used together with the most secure modes of BT and BLE (i.e., Secure Connections), allows crossing the security boundary between BT and BLE, and is even transparent to the end-user.

In this work, we present a complete description of CTKD obtained by reverse-engineering key information missing from the Bluetooth standard (i.e., CTKD negotiation and usage for BT and BLE). Then, we perform a security evaluation and we uncover four cross-transport issues (CTI) with CTKD's specification. For example, CTKD enable to (over)write and steal security keys and manipulate key authentications across BT and BLE.

We leverage the CTIs to design four novel attacks abusing CTKD capable of defeating all BT and BLE security mechanisms including Secure Simple Pairing (SSP), Secure Connections (SC), and strong associations. Our attacks enable to impersonate and take over secure sessions from any BT/BLE master or slave device. Combining master and slave impersonation the attacker can also man-in-the-middle BT and BLE secure sessions. Furthermore, a bad actor can establish secure, but unintended, BT and BLE sessions with a victim device while remaining anonymous. We name our attacks *BLUR attacks*, as they blur the security boundary between BT and BLE (by exploiting CTKD).

In contrast to prior standard-compliant attacks [2, 4, 5, 9, 20, 21, 35, 39, 40, 43, 45], our attacks are the first *cross-transport* attacks for Bluetooth as they can break BT and BLE by targeting just one of the two and the first attacks exploiting CTKD. Additionally, our attacks do not require the attacker to be present when a victim is pairing or establishing a secure session, and they result in a persistent compromise of the victim. For a more detailed comparison see Section 8.

We provide a low-cost implementation of the BLUR attack based on a Linux laptop and a Bluetooth development board. We show that the BLUR attacks are effective and standard-compliant by successfully conducting them to exploit 16 unique devices. Our set of vulnerable devices employ 14 different Bluetooth chips from Broadcom, Cambridge Silicon Radio (CSR), Cypress, Intel, Qualcomm) and covers all Bluetooth versions supporting CTKD (i.e., Bluetooth 4.2, 5.0, 5.1, and 5.2) and even a Bluetooth 4.1 device to which CTKD was backported.

We address the BLUR attacks by presenting four countermeasures addressing the four presented CTIs and the related attacks. Our mitigations can be implemented at the operating system level (i.e., Bluetooth Host) with low effort. We also evaluate one countermeasure (i.e., disable key overwriting) by implementing it on a Linux laptop. We responsible disclosed our findings to the Bluetooth SIG in May 2020. Our report is assigned with CVE-2020-15802. In September 2020, the Bluetooth SIG released a security note about our report (without contacting us) at https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/bluetooth-security/blurtooth/. We summarize our 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. Among others, CTKD enables to adversarially pair over unused transports and to tamper with BT and BLE security keys.
- Based on the identified issues we design 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.

2 BACKGROUND

We now compare the most relevant features of BT and BLE. To provide precise technical descriptions we follow the *Bluetooth standard*'s master/slave terminology instead of more apt terms like leader/follower.

2.1 A Comparison of BT and BLE

BT and BLE are two wireless transports specified in the Bluetooth standard. These transports are incompatible (e.g.,they use different physical layers and link layers) and are designed to complement each other. BT is used for high-throughput and connection-oriented services, such as streaming audio and voice. BLE is used for very low-power and low-throughput services such as localization and monitoring. 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 but different security architectures and threat models. In particular, both transports provide a pairing mechanism, named Secure Simple Pairing (SSP), to let two devices establish a shared long term key. 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 long term key while BT does not. Additionally, BT and BLE provide a secure session establishment mechanism to establish a secure communication channel using a session key derived from the long-term pairing key. During session establishment, BT allows negotiating the entropy of the session key while the BLE session key inherits the entropy of the associated long term key.

BT and BLE use the same notion of pairable and discoverable states. If a device is pairable then it will accept pairing requests from other devices. If it is discoverable it will reveal its identity when other devices scan for nearby devices. Contrary to popular belief [42], a device can answer to a pairing request even if it is not discoverable. For example, if the user knows the MAC address of her pair of headphones she can complete BT or BLE pairing from her laptop without putting the headphones into discoverable mode.

BT and BLE provide a "Secure Connections" mode which enhances their security primitives without affecting their security mechanisms. 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]. Furthermore, they provide similar ways to protect against man-in-the-middle (MitM) attacks during the pairing phase defined in the standard as association procedures. Two examples of associations are Just Works that provides no MitM protection and Numeric Comparison that provides protection against a MitM by requiring user interaction during pairing (e.g., the user has to manually confirm that she sees the same numeric code on the pairing devices).

Both BT and BLE use a master-slave medium access protocol but define the master and slave roles differently. For BT, the master is the connection initiator, the slave is the connection responder, and roles can be switched dynamically by any party after a radio link is established. For BLE, the master and slave roles are fixed and cannot be switched. The BLE master (defined as central) acts as the connection initiator and the BLE slave (defined as peripheral) as the

connection responder. 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.

3 SECURITY ANALYSIS OF CTKD

In this section, we present our security analysis of CTKD. In particular, in Section 3.1 we describe what is publicly known about CTKD, in Section 3.2 we complement it by reverse-engineering how CTKD works in practice. Finally, in Section 3.3, we present four security issues with CTKD's specification, which are the root causes of the BLUR attacks presented in Section 4.

3.1 Public Information about CTKD

Before the introduction of CTKD, a user had to pair the same two devices over BT and BLE (i.e., two times) to use both transports securely. In 2014, the Bluetooth SIG addressed this usability issue with Bluetooth 4.2 by introducing CTKD. By using CTKD, two devices, pair only once either over BT or BLE, and then can securely use both [10, p. 280]. For example, a pair of headsets and a laptop can pair over BLE, run CTKD to derive a second pairing key for BT (without the user having to put the headsets into BT pairing mode). Alternatively, the devices can pair over BT and run CTKD to generate the BLE pairing key. In both scenarios, after pairing once the headsets and the laptop can start secure sessions over BT and/or BLE.

CTKD is employed by dual-mode devices which support Secure Connections [10, p. 1401]. Those devices include laptops, headsets, tablets, smartphones, and speakers and their version is among 4.2, 5.0, 5.1, 5.2. From the Internet and our experiments we find that CTKD is supported by all major hardware and software vendors including Apple [44], Google [6], Cypress [14], Linux [13], Qualcomm [33], and Intel [22]. Notably, Apple presented it as a core and always-on Bluetooth feature during WWDC 2019. Stating at the most recent Bluetooth market update that "in 2024 all our mobile devices will be dual-mode and support CTKD" [12].

CTKD specifies a single key derivation function (KDF) based on AES-CMAC, regardless of which transport is used to pair [10, p. 1658]. The function 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, and can overwrite existing pairing keys by-design. The implementation details of CTKD's KDF are presented in Section 5.3.

Since version 5.1, the Bluetooth standard addresses a specific key overwrite attack via CTKD with the following statement: *"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]. This means that an attacker cannot overwrite a pairing key with CTKD if the overwritten key has either a lower entropy (i.e., strength) or a lower MitM protection. It is not clear why such countermeasure is enforced only for 5.1 and 5.2 devices and is not backported to all devices compatible with CTKD.

The attacks that we present in Section 4 are neither lowering the key strength or MitM requirements enforced by the standard and we experimentally validated this claim by successfully attacking 5.1 and 5.2 devices (see Section 6.2). In Section 7, we provide an extended discussion about why the key overwrite countermeasure in the standard is not effective against our attacks, and we propose effective countermeasures instead.

3.2 Reverse Engineered Details of CTKD

Since the Bluetooth standard lacks a discussion about how CTKD is negotiated and used we had to reverse-engineer (RE) these missing information. In this section we describe how CTKD works when used from BLE and BT, what we had to RE and our RE methodology. To ease our description, we abstract the protocols at a message level and we refer to the Bluetooth master and slave as Alice and Bob.

BLE pairing with CTKD. Figure 1 shows what happens when two devices are pairing over BLE and using CTKD to derive also the BT pairing key. 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 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.

BT pairing with CTKD. Figure 2 presents BT pairing with CTKD. 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}).

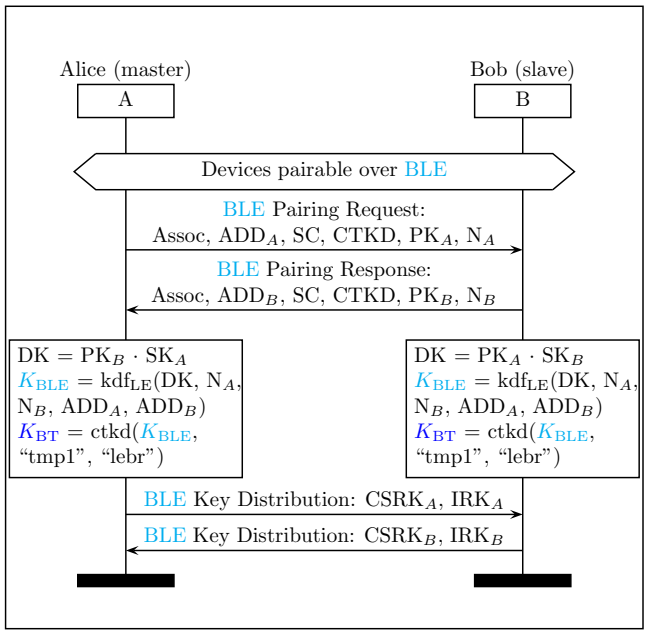

Figure 1: BLE pairing with CTKD. Alice and Bob negotiate SC and CTKD support during BLE pairing. Then, they compute the BLE pairing key and from that key, they derive the BT pairing key via CTKD (without exchanging any message over BT). Finally, they generate and exchange additional keys for BLE including 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).

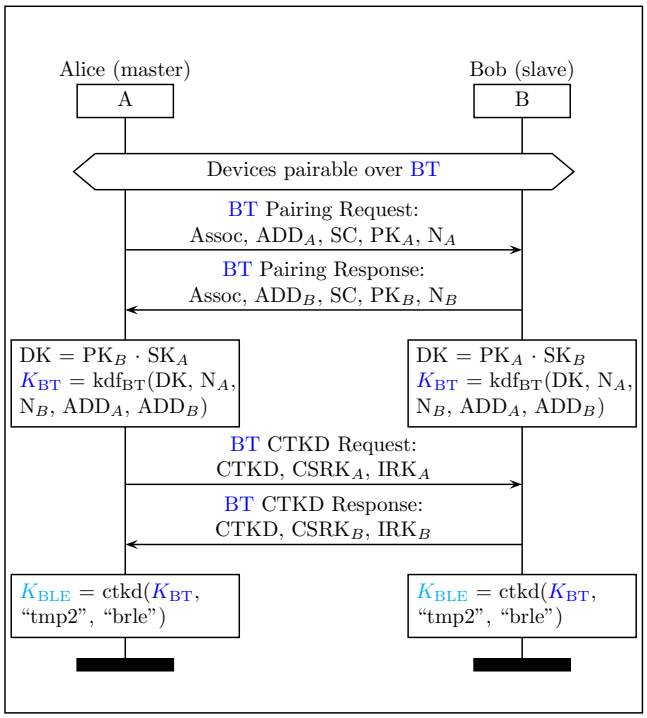

Figure 2: BT pairing with CTKD. Alice and Bob during BT pairing negotiate SC support. Then, they compute the BT pairing key, start a secure session over BT and send BT CTKD messages containing CTKD support and other keying material generated for BLE such as signature (CSRK) and identity resolving (IRK) 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).

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.

3.3 Isolated Issues with CTKD

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 [40], CTKD extends this issue across transports.

4 BLUR 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. Our attacks are stealthy as CTKD is transparent to the users, and do not require a strong attacker model as the attacker does not have to be present when the victims are pairing or establishing a secure session. As our attacks are blurring the security boundary between BT and BLE, we name them *BLUR attacks*.

The attacks were discovered by inference from the analysis presented in Section 3 and the data collected during our experiments with real devices (e.g., BT and BLE link layer and HCI packets).

4.1 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 strong association (e.g., Numeric Comparison if both have the necessary IO). This setup should protect the victims against device impersonation, traffic eavesdropping, and active man-in-the-middle attacks on BT and BLE [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 K_{BT}, a BT session key with SK_{BT}, a BLE pairing key with K_{BLE}, a BLE session key with 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.

4.2 Attacker Model and Goals

Our attacker model considers Charlie, a remote attacker who is in Bluetooth radio range with the victims. The attacker aims to compromise the secure BT and BLE sessions between the victims without tampering with their devices. The attacker's knowledge is limited to what the victims advertise over the air, e.g., full or partial Bluetooth addresses, Bluetooth names, authentication requirements, IO capabilities, and device classes.

The attacker does not know any BT or BLE key shared between the victims, does not have to be present when the victims pair or negotiate a secure session. The attacker 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.

The attacker has four goals. The first one is to impersonate Alice (to Bob) and potentially take over Alice's secure sessions. The second goal is to impersonate Bob (to Alice) and also take over Bob's secure sessions. By take over, we mean that after the attack the security bond between the two victims is broken. We note that, Alice and Bob' impersonations are different goals as they require different impersonation techniques (i.e., master and slave impersonations).

The attacker's third objective is to establish a man-in-the-middle position in a secure session between two victims and requires combining and synchronizing Alice and Bob's impersonation attacks. The fourth objective is to establish unintended and possibly stealthy sessions with Alice or Bob as an arbitrary device, without taking over a session and breaking existing security bonds. An unintended session enables the attacker to access a much broader attack surface than the one exposed in a connection-less scenario.

4.3 Attack Strategy

We now describe our attack strategy using Alice's impersonation as a reference example and 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 pretending to be Alice, and claiming to support CTKD. The attacker also declares no input/output capabilities to trigger unauthenticated JW association during pairing. This last step does not trigger the key overwrite countermeasure described in Section 3.1.

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 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 3.3 for more details). In the remaining of this section, we describe the technical details of the four BLUR attacks.

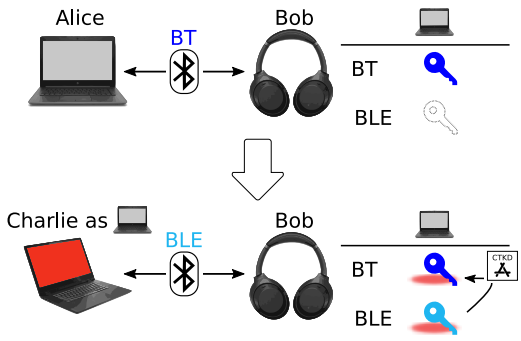

Figure 3: Attack strategy. Alice and Bob are paired over BT and run a secure BT session. Charlie pairs with Bob as Alice over BLE declaring CTKD support. Then Charlie agrees upon a BLE pairing key with Bob, and, via CTKD, tricks Bob into overwriting Alice's BT pairing key. As a result, Charlie can establish 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.

4.4 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_{BLE}, derive K_{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.

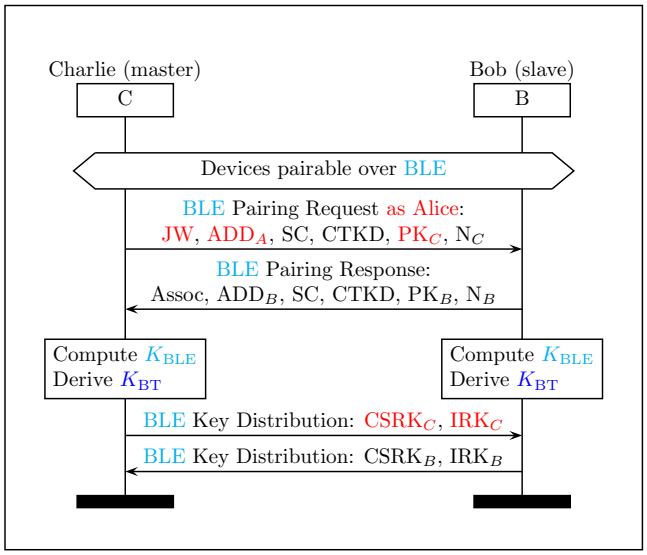

Figure 4: BLUR master impersonation attack. Charlie sends a BLE pairing request with Alice's address (ADD_A) including Just Works (JW) association, CTKD, and his public key (PK_C). Bob answers with a BLE pairing response thinking that he is talking to Alice. The attacker and the victim agree on K_{BLE}, and derive K_{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.

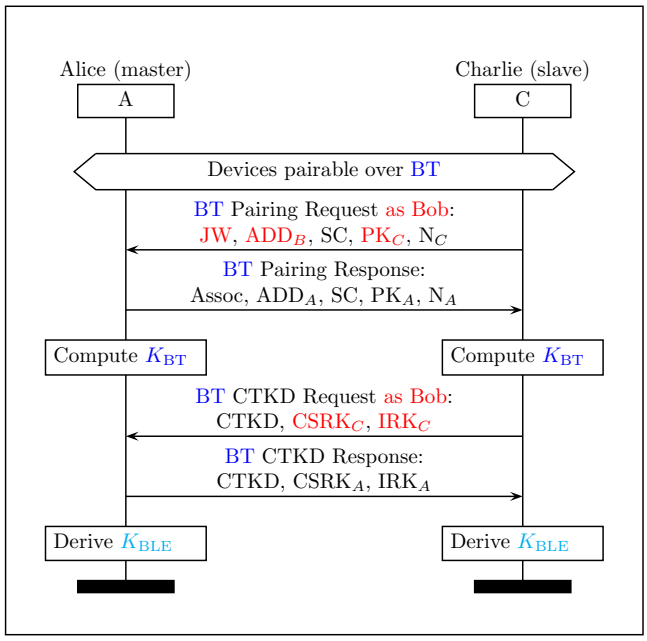

Figure 5: BLUR slave impersonation attack. Charlie sends a BT pairing request with Bob's address (ADD$_B$) including Just Works (JW) association, and his public key (PK$_C$). The pairing request is valid as BT enables to dynamically switch from slave to master before sending a pairing request. Alice answers with a BT pairing response believing that she is talking to Bob. The attacker and the victim establish K$_{BT}$, negotiate CTKD and exchange additional keying material for BLE with a BT CTKD request and response messages, and derive K$_{BLE}$. 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.

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.

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.

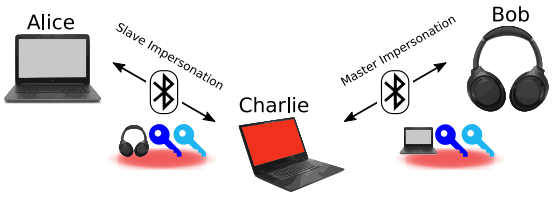

Figure 6: BLUR MitM attack. Charlie combines the master and slave impersonation attacks presented so far to establish a man-in-the-middle position between Alice and Bob both on BT and BLE.

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.

4.5 Unintended Session Attacks

The attacker can take advantage of CTKD to establish unintended secure sessions as an anonymous device. This attack is valuable for four main reasons. Firstly, the attack is stealthy as the attacker can pretend to be any device and does not have to break existing bonds. Secondly, the attacker can enumerate and tamper with all BT and BLE services running on the victim device (including the protected ones) without having to impersonate a trusted device. Thirdly, the attacker

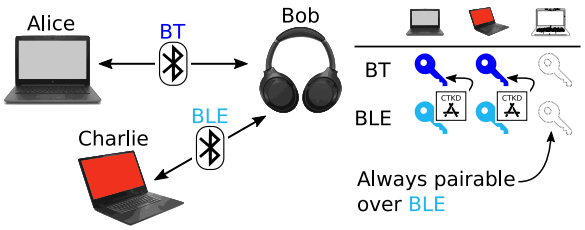

Figure 7: BLUR unintended sessions attack. Charlie can take advantage of CTKD to establish unintended BT and BLE session with Bob as a random device with arbitrary capabilities. The same can happen if Charlie targets Alice.

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

Table 1: Mapping the BLUR attacks to the CTI presented in Section 3.3. 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, an "x" if is not required and an "*" if is only required in specific cases.

can anonymously gain access to extra key material including identity resolving keys that de-anonymize BLE devices using random addresses. Finally, the attacker can silently reach more (vulnerable) code including RCE in the pairing and secure session code, which is unreachable by an untrusted device.

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). As in the impersonation attack scenario, Alice and Bob must also be pairable over BLE to support CTKD. 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.

4.6 Mapping Attacks to CTIs

Table 1 shows how the BLUR attacks take advantage of the four cross-transport vulnerabilities that we present in Section 3.3 in different ways. To cover all possible attack scenarios, a ✓ indicates that a CTI is required, an "x" if it is not required, and an "*" if it is only needed sometimes.

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.

5 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.

5.1 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 [15] and a 3rd generation ThinkPad X1 laptop as an attack device. The implementation of the attack device is presented in Section 5.2. In our evaluation, presented in Section 6, we use the same attack scenario to attack other victim devices.

Table 2 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.

5.2 Custom Attack Device

To conduct our attacks we developed a custom attack device making use of a CYW920819 development board connected to a Linux laptop (see Figure 9). Both devices BT, BLE, SC, and CTKD. Using standard laptops, smartphones or dongles is not sufficient to implement the BLUR attacks, as they do not allow to modify all device's identifiers (e.g., BT and BLE address) and all devices' capabilities advertised over the air (e.g., firmware and controller versions). A software-defined radio (SDR) is also out of scope because there is no open-source BT/BLE SDR stack currently available.

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

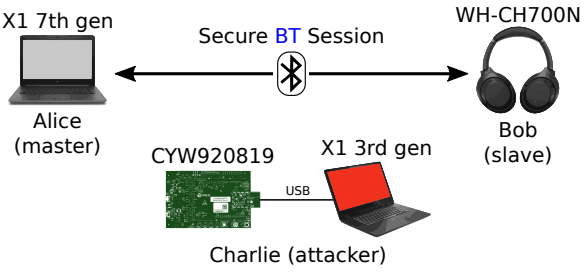

Figure 8: BLUR Attack Scenario. Alice (master) is a ThinkPad X1 7th gen, Bob (slave) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in absence of Charlie, and are running a secure BT session.

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

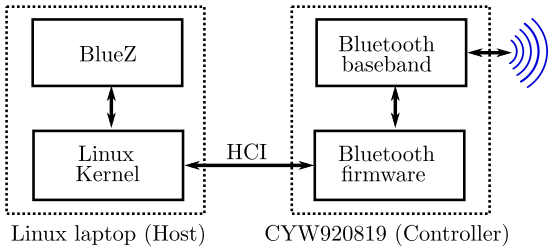

Figure 9: Attack Device Block Diagram. The attack device is composed of Linux laptop (Host) and a CYW920819 development board (Controller) connected via USB and communicating using the Host Controller Interface (HCI) protocol.

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 [29], 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,

5.3 CTKD Key Derivation Function

We implemented CTKD's key derivation function, following its specification in the Bluetooth standard [10, p. 1401]. We used our implementation to check that the keys that we observed during our experiments were correctly derived, yet, it is not required to conduct the BLUR attacks. Our implementation is written in Python 3 and uses the PyCA cryptographic module [7]. We tested it against the CTKD test vectors in the standard [10, p. 1721]. We now describe its technical details.

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

We implemented CTKD's key derivation for BT deriving and following the equation above. The key derivation computes K_{BLE} using a function $f(a, b)$ that corresponds to

AES-CMAC($key, plaintext$). If both pairing devices declare h7 support, then K_{BLE} is computed using the equation at the top otherwise the one at the bottom. h7 is a key conversion function defined in the Bluetooth standard and is negotiated during pairing using AuthReq [10, p. 1634].

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

We also implemented CTKD's key derivation for BLE deriving and following the equation above. In this case the derived key is K_{BT}. The equations' logic is identical to the one explained for BT. What changes are the input parameters. In particular, the computation uses as inputs: K_{BLE}, "tmp1", and "lebr".

6 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).

6.1 Conducting the Attacks

The BLUR attacks, presented in Section 4, 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 5.2.

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 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 K_{BLE}, derive 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 K_{BT}, derive 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.

6.2 Evaluation Results

We evaluated the BLUR attacks against 16 unique devices (employing 14 different Bluetooth chips) and our results are shown in Table 3. The 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 contain a checkmark (✓) if a device is vulnerable to master or slave impersonation attack (MI/SI), MitM, or unintended session (US) attack.

From Table 3 we confirm that the BLUR attacks are standard-compliant and very effective. All devices that we tested regardless of their implementation details are vulnerable. Moreover, all Bluetooth versions supporting CTKD are affected (i.e., Bluetooth 4.2, 5.0, 5.1, and 5.2) and the attacks are even effective on older versions of Bluetooth (e.g., 4.1 devices that backported CTKD).

Another significant conclusion that we can draw from our evaluation is that the key overwrite countermeasure in the Bluetooth standard for 5.1 and 5.2 devices [10, p. 1401] is not effective against our attacks, as a BLUR attack neither lowers key strength nor the MitM protection of the overwritten pairing key.

7 COUNTERMEASURES

In Section 3.1 we discussed the countermeasure proposed by the Bluetooth standard for 5.1 and 5.2 devices that prevents key overwrite attacks via CTKD when the overwritten key is weaker either in strength (i.e., entropy) or MitM protection. This countermeasure is not effective against the BLUR attacks as, whenever they are used to overwrite keys, they are neither downgrading the strength of nor the MitM protection of the overwritten key.

To effectively address the BLUR attacks and their root causes (CTI presented in Section 3.3) we now present four

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 3: 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.

countermeasures. Each countermeasure addresses its related CTI (e.g., C1 addresses CTI 1). Then, we describe how to implement them and how we evaluated one of them on a Linux laptop.

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 manually enable and disable pairing on a specific transport.

C2: Align BT and BLE roles. To fix role asymmetries between BT and BLE when using CTKD, a device should store the transport and the role used while pairing and enforce it across re-pairings regardless of the transport in use. 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 while pairing if a trusted pairing key already exists for the other transport. As a result, to overwrite a trusted 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 pairing key (that would be used as input to CTKD). A key is considered 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 the association mechanisms used when pairing on different transports, a device should keep track of the association mechanism used while pairing for the first time with a device and enforce it for subsequent re-pairings across BT and BLE. There is no obvious 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 not only address the four CTIs but they also stop the BLUR attacks. In particular, C3 prevents impersonation and MitM as the attacker will not be able to write and overwrite key across transports but only target separately BT and BLE. To stop the unintended sessions attacks C1 is also needed as the attacker should not be able to pair with CTKD on unused transports. C2 and C4 help to mitigate the attacks by providing more defense-in-depth but they are not strictly required.

Our countermeasures can be implemented in the Bluetooth Host component (i.e., device's main OS). C2, C3, and C4 can be realized by keeping track of metadata that is already

exchanged during the pairing protocol (e.g., device role, association) and aborting the protocol when needed. C1 can be implemented with a timer which disables pairability on a transport when not needed and a simple user interface to monitor and switch on/off pairability for BT and BLE.

To verify the effectiveness of C3 we implemented a C3 proof-of-concept and tested it using a Linux laptop. We paired our laptop with the victim device using CTKD and we deleted the pairing data on the victim device and then used it as the attacker device. Then, to disable CTKD on the laptop, we unset the write permission bit in the folder and the file storing the pairing keys. Then we ran the impersonation attack from the attack device and the attack failed as the OS was preventing the Bluetooth Host from (over)writing new pairing keys.

8 RELATED WORK

Bluetooth provides a royalty-free and widely-available cable replacement technology [19]. Bluetooth standard compliant attacks are particularly dangerous as all Bluetooth devices are affected, regardless of version numbers or implementation details. Such standard-compliant attacks have appeared since the first versions of Bluetooth [23, 28]. Standard-compliant attacks on BT include attacks on legacy pairing [38], secure simple pairing (SSP) [9, 20, 39], Bluetooth association [21, 40], key negotiation [2], and authentication procedures [4, 27, 41]. Standard-compliant attacks on BLE include attacks on legacy pairing [35], key negotiation [5], SSP [9, 45], reconnections [43], and GATT [24]. Compared to the mentioned attacks that target either BT or BLE, the BLUR attacks are the first standard-compliant attacks targeting the intersection between BT and BLE.

We have seen attacks targeting specific implementation flaws on BT [36] and BLE [18, 37]. As our BLUR attacks target the specification level, they are effective regardless of the implementation details. Several surveys on BT and BLE security were published [16, 30, 32] 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 threat model.

Cross-transport 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]. Our BLUR attacks are the first cross-transport attacks for BT and BLE.

The cryptographic primitives used by Bluetooth have been extensively analyzed. For example, the E_0 cipher used by BT was investigated [17] and it is considered relatively weak [32]. SAFER+, used for authentication, was analyzed as well [26]. BT and BLE "Secure Connections" use the AES-CCM authenticated-encryption cipher. AES-CCM was extensively analyzed [25, 34] and it is FIPS-compliant. Our BLUR attacks target key negotiation and not cryptographic primitives, and are effective even with perfectly secure cryptographic primitives.

As can be seen from Table 4, compared to other standard-compliant attacks, the BLUR attacks are novel and are enabling impactful attack scenarios. The BLUR attacks are the first cross-transport attacks for Bluetooth and are the first attacks exploiting CTKD. In terms of impact, the BLUR attacks require a weak attacker model as the attacker does not have to observe previous pairing and secure sessions between the victim. On top of that, they break even the most secure BT and BLE mode (i.e., SSP, LESC, SC, and strong association) and their effect is persistent.

9 CONCLUSION

In this work we examine CTKD, a usability feature in the Bluetooth standard that has, until now, not been scrutinized for security issues by the research community. We develop four attacks that take advantage of CTKD to exploit both BT and BLE. Our attacks are the first examples of cross-transport attacks on Bluetooth, they are standard-compliant, and effective against the most secure BT and BLE modes (i.e., Secure Connections and Secure Connections Only). Our attacks are the first ones that achieve a persistent compromise of the devices, i.e., it leaves the devices in a compromised state even when the attacker is no longer present. In contrast to other prior standard-compliant attacks (i.e., attacks that also are not targeting implementation bugs), our attacks are not limited to the pairing phase. That means we can execute the attack on any device at any time, without forcing a new pairing event.

With our BLUR attacks we reach four significant goals. We achieve impersonation and take-over for both the master and slave devices; man-in-the-middle on secure sessions in the most secure mode (Secure Connections); and establishing unintended sessions as an anonymous device. Collectively our attacks are called BLUR attacks as they blur the security boundary between BT and BLE.

To demonstrate the practicality of the BLUR attacks, we presented a low-cost implementation based on cheap readily available hardware (a laptop, and a Bluetooth development board) and open-source software (Linux, and internalblue). We also describe solutions to the main technical challenges we faced during development, including low-level modifications of a Bluetooth firmware.

We use our implementation to experimentally confirm that CTKD-compatible devices (using 14 unique Bluetooth chips) are vulnerable in practice. Our attacks are successful on all the devices we tested which shows that this is a serious problem in practice. We end the paper by discussing the feasibility of various low-cost, host-based countermeasures that prevent the attacks at the cost of some usability. We followed a responsible disclosure process and notified the Bluetooth SIG of our findings, resulting in CVE-2020-15802, and we intend to release our attack implementation as an open source project.

REFERENCES

[1] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. 2016. Practical bluetooth traffic sniffing: Systems and privacy implications. In

Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services. ACM, 333–345.
[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation of Bluetooth BR/EDR. In *Proceedings of the USENIX Security Symposium*. USENIX.
[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2019. Nearby Threats: Reversing, Analyzing, and Attacking Google's "Nearby Connections" on Android. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.
[4] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. BIAS: Bluetooth Impersonation AttackS. In *Proceedings of Symposium on Security and Privacy (S&P)*. IEEE.
[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *Transactions on Privacy and Security (TOPS)* (2020). https://doi.org/10.1145/3394497
[6] AOSP. 2020. Fluoride Bluetooth stack. https://chromium.googlesource.com/aosp/platform/system/bt/+/master/README.md, Accessed: 2020-01-27. (2020).
[7] Python Cryptographic Authority. 2019. 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. 2016. Staying secure and unprepared: Understanding and mitigating the security risks of apple zeroconf. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 655–674.
[9] Eli Biham and Lior Neumann. 2018. 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. 2019. Bluetooth Core Specification v5.2. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726, Accessed: 2020-01-27. (2019).
[11] Bluetooth SIG. 2019. Bluetooth Markets. https://www.bluetooth.com/markets/. (2019).
[12] Bluetooth SIG. 2020. Bluetooth Market Update 2020. https://www.bluetooth.com/bluetooth-resources/2020-bmu/. (2020).
[13] BlueZ. 2014. Bluetooth 4.2 features going to the 3.19 kernel release. https://tinyurl.com/q9dzh2h, Accessed: 2020-01-27. (2014).
[14] Cypress. 2019. BLE and Bluetooth. https://www.cypress.com/products/ble-bluetooth, Accessed: 2020-01-27. (2019).
[15] Cypress. 2019. CYW920819EVB-02 Evaluation Kit. https://www.cypress.com/documentation/development-kitsboards/cyw920819evb-02-evaluation-kit, Accessed: 2019-11-16. (2019).
[16] John Dunning. 2010. Taming the blue beast: A survey of Bluetooth based threats. *IEEE Security & Privacy* 8, 2 (2010), 20–27.
[17] Scott Fluhrer and Stefan Lucks. 2001. Analysis of the E0 encryption system. In *Proceedings of the International Workshop on Selected Areas in Cryptography*. Springer, 38–48.
[18] Garbelini, Matheus and Chattopadhyay, Sudipta and Wang, Chundong. 2020. SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. https://asset-group.github.io/disclosures/sweyntooth/sweyntooth.pdf, Accessed: 2020-04-08. (2020).
[19] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf J Joeressen, and Warren Allen. 1998. Bluetooth: Vision, goals, and architecture. *ACM SIGMOBILE Mobile Computing and Communications Review* 2, 4 (1998), 38–45.
[20] Keijo Haataja and Pekka Toivanen. 2010. Two practical man-in-the-middle attacks on Bluetooth secure simple pairing and countermeasures. *Transactions on Wireless Communications* 9, 1 (2010), 384–392.
[21] Konstantin Hypponen and Keijo MJ Haataja. 2007. "Nino" man-in-the-middle attack on bluetooth secure simple pairing. In *Proceedings of the International Conference in Central Asia on Internet*. IEEE, 1–5.
[22] Intel. 2019. Intel Wireless Solutions. https://www.intel.com/content/www/us/en/products/wireless.html, Accessed: 2020-01-27. (2019).
[23] Markus Jakobsson and Susanne Wetzel. 2001. Security weaknesses in Bluetooth. In *Proceedings of the Cryptographers' Track at the RSA Conference*. Springer, 176–191.
[24] Sławomir Jasek. 2016. Gattacking Bluetooth smart devices. Black Hat USA Conference. (2016).
[25] Jakob Jonsson. 2002. On the security of CTR+ CBC-MAC. In *Proceedings of the International Workshop on Selected Areas in Cryptography*. Springer, 76–93.
[26] John Kelsey, Bruce Schneier, and David Wagner. 1999. Key schedule weaknesses in SAFER+. In *Proceedings of the Advanced Encryption Standard Candidate Conference*. NIST, 155–167.
[27] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M Ufuk Çağlayan. 2004. Relay attacks on Bluetooth authentication and solutions. In *Proceedings International Symposium on Computer and Information Sciences*. Springer, 278–288.
[28] Andrew Y Lindell. 2008. Attacks on the pairing protocol of Bluetooth v2.1. *Black Hat USA, Las Vegas, Nevada* (2008).
[29] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. 2019. InternalBlue - Bluetooth Binary Patching and Experimentation Framework. In *Proceedings of Conference on Mobile Systems, Applications and Services (MobiSys)*. ACM.
[30] Nateq Be-Nazir Ibn Minar and Mohammed Tarique. 2012. Bluetooth security threats and solutions: a survey. *International Journal of Distributed and Parallel Systems* 3, 1 (2012), 127.
[31] Michael Ossmann. 2019. Project Ubertooth. https://github.com/greatscottgadgets/ubertooth, Accessed: 2019-10-21. (2019).
[32] John Padgette. 2017. Guide to bluetooth security. *NIST Special Publication* 800 (2017), 121.
[33] Qualcomm. 2019. Expand the potential of Bluetooth. https://www.qualcomm.com/products/bluetooth, Accessed: 2020-01-27. (2019).
[34] Phillip Rogaway. 2011. Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan* (2011).
[35] Mike Ryan. 2013. Bluetooth: With Low Energy Comes Low Security. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, Vol. 13. USENIX, 4–4.
[36] Ben Seri and Gregory Vishnepolsky. 2017. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/, Accessed: 2018-01-26. (2017).
[37] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. 2019. BLEEDINGBIT: The hidden Attack Surface within BLE chips. https://armis.com/bleedingbit/, Accessed: 2019-07-24. (2019).
[38] Yaniv Shaked and Avishai Wool. 2005. Cracking the Bluetooth PIN. In *Proceedings of the conference on Mobile systems, applications, and services (MobiSys)*. ACM, 39–50.
[39] Da-Zhi Sun, Yi Mu, and Willy Susilo. 2018. Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard V5. 0 and its countermeasure. *Personal and Ubiquitous Computing* 22, 1 (2018), 55–67.
[40] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. 2021. Method Confusion Attack on Bluetooth Pairing. In *Proceedings of Symposium on Security and Privacy (S&P)*. IEEE.
[41] Ford-Long Wong, Frank Stajano, and Jolyon Clulow. 2005. Repairing the Bluetooth pairing protocol. In *Proceedings of International Workshop on Security Protocols*. Springer, 31–45.
[42] Joshua Wright. 2018. I Can Hear You Now - Eavesdropping on Bluetooth Headsets. https://www.willhackforsushi.com/presentations/icanhearyounow-sansns2007.pdf, Accessed: 2018-10-30. (2018).
[43] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. 2020. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *14th USENIX Workshop on Offensive Technologies (WOOT)*.
[44] Apple WWDC. 2019. What's New in Core Bluetooth. https://developer.apple.com/videos/play/wwdc2019/901, Accessed: 2020-01-27. (2019).
[45] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *29th USENIX Security Symposium (USENIX Security 20)*. 37–54.

10 APPENDIX

Lessons Learned

In this section we list the main lessons that we learned in the hope that they will be useful for protocol designers who are dealing with cross-transport security mechanisms.

			Attack				
Year	Paper	Target	Phase	C I AK	SC/SCO	Persistent	Note
Attacks on BT							
2016	Albazrqaoe et al. [1]	Standard	Any	◐○○○	x	-	BlueEar Sniffer
2017	Seri et al. [36]	Impl.	Pairing	●●●○	NA	✓	BlueBorne
2018	Sun et al. [39]	Standard	Pairing	●●●○	✓	-	Passkey (MitM)
2018	Biham et al. [9]	Impl.	Pairing	●●●◐	NA	✓	Fixed Coordinate Invalid Curve
2019	Antonioli et al. [2]	Standard	Pairing	●●◐○	✓	-	KNOB (MitM)
2020	Antonioli et al. [4]	Standard	Pairing	●●●○	✓	-	BIAS
2021	Tschirschnitz et al. [40]	Standard	Pairing	●●●○	✓	-	Method Confusion (MitM)
Attacks on BLE							
2016	Jasek et al. [24]	Standard	NA	◐○○○	x	-	Black Hat
2019	Seri et al. [37]	Impl.	NA	○◐◐○	NA	✓	Bleedingbit
2020	Zhang et al. [45]	Standard	Pairing	◐◐◐○	✓	-	MitM (SCO)
2020	Wu et al. [43]	Standard	Session	○○●○	✓	-	BLESA
2020	Garbelini et al. [18]	Impl.	Any	◐◐◐○	NA	-	SweynTooth fuzzer
Attacks on both BLE and BT							
2019	Ossmann et al. [31]	Standard	NA	◐○○○	x	-	Ubertooth sniffer
2020	Antonioli et al. [5]	Standard	Pairing	●●◐○	✓	-	Downgrade (MitM)
2021	**This work**	Standard	Any	●●●◐	✓	✓	BLUR (cross-transport)

Table 4: Overview of recent attacks on Bluetooth and BLE. C = Data Confidentiality, I = Data Integrity, A = Device Authentication, K = Key disclosure. No (○) Partially (◐), Yes (●).

Cross-transport 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.

Cross-transport 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 the same 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 overweight the former.