Paper changes for CCS21 compared to SEC21

In this document, we summarize our changes concerning the last Major Revision that we got from USENIX Security that after the discussion ended up with a Reject and Resubmit decision. The USENIX Security reviewers asked for a complete re-write of several sections of the paper and that is what we did. In the following subsection we describe our modifications in detail. For reference, we are also attaching the last reviews and a paper diff.

Description of CTKD in non-adversarial settings (Sections 2, 3)

We rewrote Section 2 and 3 to clarify what information about CTKD is present in the Bluetooth standard and other public information (see 3.1) and what we had to experimentally reverse-engineer (see 3.2). Doing so, we are also clarifying our contribution to the analysis of CTKD. Leveraging what we reverse-engineered, we now provide dedicated descriptions with related Figures of how CTKD works for BT and BLE, and we emphasize their differences.

Section 3 now expresses our contribution related to CTKD, provides an introduction to the BLUR attacks, enables a reader to tinker with CTKD and come up with their own attacks, and also serve as future documentation for CTKD which was so far lacking (even in the Bluetooth standard). Finally, as requested, we thought again about the attack root causes, we reduced them from five to four and we moved them in the discussion section (Section 7).

Presentation of the BLUR attacks (Section 4)

We worked on Section 4 to simplify the presentation of our attacks. In particular, we introduce the attacks by first presenting a high-level attack strategy in 4.3 that explains the attacker's tricks. Then, we improve the technical description of each presented attack both in text and visually by coloring in red the fields modified by the attacker. Overall Section 4 should be self-contained and use the right level of abstraction to let the reader appreciate attacks and their novelty. For example, we want to make clear that is not only yet another attack abusing "Just Works".

Re-implementation of CTKD's derivation function (Section 5)

We worked on Section 5 to clarify our re-implementation of the CTKD key derivation function. In 5.3 we clarify that what we are not re-implementing the whole CTKD protocol but focus on its key derivation function and that our implementation is not required to conduct the attack. Moreover, we also rewrote 5.2 to better explain why we needed to build our attack device and

what its capabilities are compared to standard laptops, smartphones, and even software-defined radios.

Attacks' root causes and countermeasures (Section 7)

We improve our analysis of the BLUR attacks in Section 7. In 7.1 we re-wrote our description of the presented cross-transport issues (CTI) and we clearly state these CTIs represent the attacks' root causes and that were extrapolated from the BLUR attacks. We also address the reviewers' concerns with the proposed countermeasures by rewriting 7.2 such that it is clear which Countermeasure is needed to fix which attack and/or cross-transport issue. We rewrote 7.3 to better distill our lesson learned. Regaling why the presented issues are in the standard, we cannot comment as we are not part of the Bluetooth SIG.

Attacks' comparison with related work (Section 8)

We were asked to place the BLUR attacks in the context of prior related work. To address this comment, we added Table 3 which compares the BLUR attacks with state-of-the-art attacks on BT or BLE. With the help of Table 3 we want to clearly communicate what is the role of the BLUR attacks withing other standard-compliant and implementation-specific examples. It should be now clear that the BLUR attacks reach unprecedented goals such as persistence, and effectiveness regardless of the victim's state and targeted transport.

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

Anonymous Author(s)

ABSTRACT

The Bluetooth standard specifies two incompatible wireless transports: Bluetooth Classic (BT) ~~for high-throughput services~~ and Bluetooth Low Energy (BLE)~~for very low-power services. Despite the similarity in name and use of similar security mechanisms, BT and BLE~~ . The two transports have different security architectures and threat models ~~. In particular, pairing enables two devices to establish a long-term key to secure the communication. Two devices~~ and provide dedicated pairing protocols to establish long-term keys. Traditionally, two devices would have to pair over BT and BLE to use both ~~transports securely. Since pairing the same devices twice is considered "user-unfriendly"~~securely. But in 2014, Bluetooth v4.2 ~~introduced Cross-Transport Key Derivation (CTKD)~~addressed this usability issue by introducing *Cross-Transport Key Derivation (CTKD)* for BT and BLE. CTKD allows ~~two devices to pair once, either over BT or BLE, and generate both BT and BLE long-term keys . Despite CTKD allowing traversal of~~ establishing BT and BLE pairing keys just by pairing over one transport. Despite the fact that CTKD crosses the security boundary between BT and BLE, the ~~security implications of CTKD have not yet been investigated~~Bluetooth standard does not include CTKD in its threat model and does not provide a complete description of it.

~~We present the first~~ To address these issues, we present a full characterization of CTKD obtained via reverse-engineering and a security analysis of CTKD~~and identify~~ . Based on our findings we introduce four standard-compliant attacks on CTKD breaking the strongest BT and BLE security modes. Our attacks are the first examples of cross-transport ~~issues at the Bluetooth specification level. These issues enable, for the first time, exploitation of both~~ attacks for Bluetooth, as they enable breaking BT and BLE by ~~attacking either transport. Based on the identified issues, we demonstrate four novel cross-transport attacks resulting in deviceimpersonation, traffic manipulation, and malicious session establishment~~targeting just one of the two. In contrast to prior standard-compliant attacks, our attacks do not require the attacker to be present when the victims are paring or establishing secure sessions, and their effect is persistent. We describe how the attacks can be used to impersonate and take over any device, man-in-the-middle secure sessions, and establish unintended sessions as an anonymous device. We refer to ~~them as BLUR attacks~~ our attacks as *BLUR attacks*, as they blur the security boundary between BT and BLE. ~~The BLUR attacks are standard-compliant and therefore apply to all devices supporting CTKD, regardless of implementation. We successfully demonstrate~~ We provide a low-cost implementation of the BLUR attacks and we successfully evaluate them on 13 devices with 10 unique Bluetooth chips ~~, and discuss effective countermeasures~~from popular vendors such as Cypress, Dell, Google, Lenovo, Samsung, and Sony. 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.~~ 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 [**?**]. Bluetooth is specified in an open standard maintained by the Bluetooth special interest group (SIG)~~. The latest version of the standard~~ , and its latest version is 5.2 [**?**]. The standard specifies two ~~different, incompatible wireless transports,~~ 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 best suited for very low-power~~ . While BLE is optimized for connection-less and very-low-power use cases such as localization and ~~monitoring~~digital contact-tracing.

~~As BT and BLE were introduced at different points in time to address different use cases, the standard maintains *separate security architectures and threat models*~~ The Bluetooth standard defines different security architectures and threat models for BT [**?** , p. 947] and BLE [**?** , p. 1617]. ~~While these security architectures address different threat models, they use similar security mechanisms, including~~ Both transports provide pairing and secure session establishment protocols. Pairing enables ~~devices to establish a~~ the establishment of shared long term ~~key~~keys, and secure session establishment ~~enables~~ allows paired devices to ~~establish a secure communication channel by negotiating a session key that is~~ create a secure channel through a (fresh) session key derived from the pairing ~~long term~~ key.

~~Devices that support both BT and BLE~~ Traditionally, two devices would have to pair ~~twice to use both transports securely.~~ over BT *and* BLE to securely use both. However, pairing the same devices two times is considered user-unfriendly. To address this usability issue, Bluetooth v4.2 ~~(released in 2014)~~ introduced *Cross-Transport Key Derivation (CTKD)* ~~to mitigate the "user-unfriendly" requirement to pair the same devices twice. After pairing on one transport, CTKD allows the creation of a second long term key for the other transport~~for BT and BLE in 2014. CTKD enables to pair two devices once, either on BT or BLE, and negotiate BT and BLE pairing keys without having to pair a second time [**?** , p. 1401]. For example, two devices can pair over ~~BT, generate the BT long term~~ BLE declaring CTKD support, agree on a BLE pairing key, and ~~then run CTKD to derive the BLE long term key (without having to pair over BLE)~~derive a BT pairing key without using BT. Alternatively, they can use CTKD from BT to derive BT and BLE pairing keys. All major Bluetooth software ~~stacks (~~(e.g., Apple, Linux, Android, and Windows) and hardware providers (e.g., Cypress, Intel, Qualcomm, Broadcom, Apple, Sony, and Bose) ~~implement CTKD.~~ support CTKD.

Actually, Apple presented CTKD as a core ~~"always on" Bluetooth~~ and always-on feature to improve Bluetooth's usability [?].

~~We present the first security analysis of CTKD, uncovering~~ Security-wise, CTKD has not received any attention from the research community and is only partially documented in the Bluetooth standard. In particular, CTKD is not part of the Bluetooth threat model and the standard does not provide a complete description of it. On the other hand, CTKD is a very interesting, yet-unexplored, attack surface, as it is a standard-compliant ~~security issues. Those issues are the first examples of cross-transport vulnerabilities for Bluetooth~~ feature, is used together with the most secure modes of BT and BLE (i.e., Secure Connections), is crossing the security boundary between BT and BLE, and is transparent to the end-user.

In our work, we provide a complete description of CTKD obtained by merging the scattered and incomplete information about CTKD from the Bluetooth standard, and the result of reverse-engineering experiments conducted with actual devices. Based on our ~~findings, we demonstrate four cross-transport attacks, enabling device impersonation, traffic interception, and traffic manipulation, as well as unintended device sessions~~description, we performed a security evaluation of CTKD and we present four novel and standard-compliant attacks on CTKD. Our attacks ~~enable BT and BLE cross-transport exploitation , are standard-compliant and likely affect all devices supporting CTKD. We name our attacks BLUR attacks, as by exploiting CTKD they blur the security boundary between~~ are the first examples of *cross-transport* exploitation for Bluetooth, as they exploit BT and BLE only by targeting one transport.

The attacks are very effective as they can defeat all BT and BLE ~~.~~

security mechanisms including Secure Simple Pairing (SSP), Secure Connections (SC), and strong associations. In contrast to ~~previously published attackson BT and BLE~~prior standard-compliant attacks [? ? ? ? ? ? ? ? ? ? ?], our attacks do not require the attacker to be present during pairing ~~or~~ and secure session establishment ~~. Therefore, our attacks have lower requirements for the attacker while still breaking~~ and they result in a persistent compromise of the victims.

Using our attacks we can reach several high-impact goals. In particular, they enable to impersonate and take over secure sessions from any BT or BLE device, man-in-the-middle BT and BLE secure sessions, and establish unintended BT and BLE sessions with a victim device while remaining anonymous and without breaking existing security bonds. We name our attacks *BLUR attacks*, as they blur the security boundary between BT and BLE~~security guarantees~~.

~~We implement the BLUR attacks using a widely available~~ We provide a low-cost implementation of the BLUR attack based on a Linux laptop and a Bluetooth development board~~connected to a laptop running Linux and developing custom software based on open-source tools. This makes reproducing~~ . We show that the BLUR attacks ~~simple and affordable. Our evaluation demonstrates that all tested devicesare vulnerable. We will release our tools to the public after the responsible disclosure process completes. We use our attack implementation to evaluate~~ are a real and standard-compliant threat by successfully conducting them on a diverse set of devices. In particular, we use our implementation to exploit 13 ~~devices, with~~ unique devices employing 10 unique Bluetooth chips ~~, from the~~ from major hardware and software vendors ~~, e.g.,~~(i.e., Broadcom, Cambridge Silicon Radio~~(CSR)~~, Cypress, Google, Intel, Linux, Qualcomm, and Windows~~and representing all Bluetooth versions that support~~) implementing the most common Bluetooth versions supporting CTKD (i.e., Bluetooth versions 4.1, 4.2, 5.0, and 5.1)~~and even a device supporting Bluetooth version 4.1 to which CTKD has been backported.~~ .

To concretely address the presented attacks we infer their root causes by listing four cross-transport issues with the specification of CTKD. Then, we address those issues and the related BLUR attacks by proposing four effective countermeasures that can be implemented at the operating-system level (i.e., in the Bluetooth Host) with low effort. We summarize our contributions as follows:

- We ~~perform the~~ reverse-engineered CTKD and performed its first security analysis~~of CTKD(Section 3), and show that it enables crossing the security boundary between BT and BLE. We identify novel and very serious issues, enabling~~ . Based on that, we design four standard-compliant attacks on CTKD. The attacks break all BT and BLE security mechanisms including SSP, SC, and strong association, do not require the attacker to be present while the victims are pairing and establishing secure sessions, and their effect is persistent. Moreover, our attacks are the first examples of cross-transport attacks ~~between~~ for Bluetooth as they exploit BT and BLE by targeting either of the two.
- ~~We propose four attacks to exploit the issues in CTKD (Section 4).~~ Our attacks ~~allow impersonation , interception, traffic manipulation, and unintended sessions. In Section 5, we~~ result in impersonation and take over of devices, MitM their secure sessions, and establishment of unintended sessions as an anonymous device. 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 confirm that real-world BT and BLE~~ We use our implementation to confirm that actual devices are vulnerable to the BLUR attacks by ~~evaluating our attacks on~~ successfully attacking 13 ~~unique devices (Section 6~~different devices employing 10 unique Bluetooth chips and covering the majority of Bluetooth versions compatible with CTKD (e.g., 4.1, 4.2, 5.0, and 5.1). We ~~provide concrete~~ discuss four concrete attacks' root causes in the specification of CTKD and we provide four practical countermeasures to fix ~~the presented issues.~~ them.
- We disclosed our findings and countermeasures to the Bluetooth SIG in May 2020. The Bluetooth SIG acknowledged ~~our findings~~ them and assigned CVE-2020-15802 to the BLUR attacks. In September 2020, the Bluetooth SIG released a security note about our report ~~at~~ (without contacting us) at https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/bluetooth-security/blurtooth/.

2 BACKGROUND

We now compare the most relevant features of BT and BLE~~, and introduce CTKD.~~ . 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 (i.e., while they use the same 2.4 GHz band the physical e.g., they use different physical layers and link layers are different) 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. Typically, high-end High-end devices, such as laptops, smartphones, headsets, and tablets, provide both BT and BLE (in a single radio chip), 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. Both 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) [? , p. 1666] , while BT SSP uses the Link Manager Protocol (LMP) [? , p. 568] . During pairing, BLE allows negotiating the entropy of the long term key while BT does not. Both transports Additionally, BT and BLE provide a secure session establishment mechanism to derive a session key from the long term keyand protect the communicationestablish over BLE BLE (i.e., two times) 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 BLE the BLE session key inherits the entropy of the session keyfrom the entropy of the long term keyassociated 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 [?], 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 support provide a "Secure Connections" mode that uses FIPS compliant security primitives such as AES-CCM for authenticated encryption, Elliptic-Curve Diffie-Hellman (ECDH) over P-256 for key agreement, mutual authentication procedures for the long-term key, and AES-CMAC for keyed-hashing. BT and BLE have similar association procedures that can be used to protect the pairing phase against . Furthermore, they provide similar ways to protect against man-in-the-middle attacks(MitM) attacks during the pairing phase defined in the standard as association procedures. Two examples of associations are "Just Works " Just Works that provides no protection and "Numeric Comparison " MitM protection and Numeric Comparison that provides protection against man-in-the-middle attacks 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 define use a master-slave medium access protocol but define the master and slave roles in different waysdifferently. For BT, the master is the connection initiator, the slave is the connection responder, and roles can be switched . Both master or slave can request a role switch almost anytime dynamically by any party after a radio link between the two is established. For BLE, the master and slave roles are fixed and switching roles is not supported. The master cannot be switched. The BLE master (defined as central) acts as the connection initiator (BLE central) and the slave as and the BLE slave (defined as peripheral) as the connection responder(BLE peripheral). High-end BLE devices, such as laptops and smartphones, implement both master and support both BLE master and BLE slave modes and are typically used as the masterBLE masters, while low-end devices, such as fitness trackers or smartwatches, typically implement only the 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, and in Section 3.2 we complement it by reverse-engineering how CTKD works in practice for BT and BLE.

3.1 Public Information about CTKD

Two devices that support BT and BLE have to pair 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. Pairing the same two devices twice is considered "The Bluetooth SIG considered this procedure user-unfriendly " and the Bluetoothstandard version and improved Bluetooth's usability by introducing CTKD for Bluetooth 4.2(released in 2014) introduces CTKDto address this issue. As shown in Figure ??, CTKD enables two devicesto pair once, in 2014. By using CTKD, two devices, pair only one time either over BT or BLE, and then can securely use both [? , p. 280]. For example, a user can pair a headset pair of headsets and a laptop can pair over BLE, without putting the headset in BT discoverable mode, and then securely connect the headset and the laptop over BT (without having to pair over BT). It is also possible to do the initial pairing over BT , and use 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.

Before explaining CTKD, it is important to review the differences between *pairable (bondable)* and *discoverable* states for BT andBLE

The Bluetooth standard specifies that CTKD should be used only when a device supports Secure Connections mode for that specific transport [? , p. 1401]. Secure Connections is a security mode that was introduced both for BT and BLE to enhance 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 [? , p. 269]. As a consequence, an attacker who can break CTKD can break BT and BLE's strongest security mode. If a device is *pairable* then it will accept pairing requests from other devices. If if it is *discoverable* it will reveal its identity when other devices scan for nearby devices. Contrary to popular belief, a device is not

required to be both discoverable and pairable for pairing but it only needs to be pairable. The device that initiates pairing only needs to know the identity (MAC address) of the pairable target device. For example, when pairing a laptop with headphones over BT, typically only the headphones are discoverable and pairable while the laptop is only pairable. Hence, it is possible to pair with a device even if it is not discoverable [?].

CTKD overview. CTKD is used by two devices who paired and share a long term key over BLE to derive a long term key for BT. CTKD can also be used to derive BLE pairing keys after two devices paired over BT.

The Bluetooth standard specifies the same CTKD function to derive also describes how CTKD derives pairing keys for BT and BLE [? , p. 1658]. CTKD uses the same key derivation function for BT and BLElong term keys. This , and the function takes as inputs a 128-bit (16-byte) key and two 4-byte strings and derives a 128-bit (16-byte) keyusing AES-CMAC (see Section 5.3 for CTKD's internals). CTKD for BT derives a BLE long term . What changes between BT and BLE are the strings used as inputs. When CTKD is used to derive a BLE pairing key (K_{BLE}) from a BT long term pairing key (K_{BT}) and the strings "tmp2" and "brle", while CTKD for BLE derives from and the strings "tmp1" and "lebr". As the standard defines constant strings and no fresh nonces as inputs, the CTKD function derives the same output key when reusing the same input then the key is derived using the "tmp2" and "brle" strings. In the other case, the derivation is performed using the "tmp1" and "lebr" strings. We note that the CTKD key derivation function is *deterministic*, as using CTKD on the same input key will always generate the same output key.

CTKD is broadly supported by , e.g., Despite being an optional feature, from the Internet and our experiments we can conclude that CTKD is supported by all major hardware and software vendors including Apple [?], Google [?], Cypress [?], Linux [?], Qualcomm [?], and Intel [?]. CTKD is combined with "Secure Connections ", a security mode that was introduced to enhance the security primitives of BT and BLE without affecting their security mechanisms. For example , "Secure Connections" introduces AES-CCM authenticated encryption for BT , and ECDH pairing for BLE. Actually, Apple presented it as a core and always-on Bluetooth feature during WWDC 2019.

3.2 Reverse Engineered Details on CTKD

The Bluetooth standard lacks a section about CTKD negotiation and usage for BT and BLE, but merely provides scattered information. Since knowing such information is essential to perform our security analysis, we reverse-engineered it. In this section we provide an high-level summary of the information that we extracted from our reverse-engineering process. To ease our description we abstract the protocols at a message level, where each message captures one or more packets sent over the air. Furthermore, we refer to the Bluetooth master as Alice, and the Bluetooth slave as Bob.

CTKD from BLE. Figure 1 shows CTKD during BLE pairing. Alice and Bob are pairable BLE and discover each other using BLE's advertising and scanning features. Then, Alice and Bob negotiate specific capabilities using pairing request and response messages. The messages must contain Secure Connections (SC) and CTKD support, together with an association method (Assoc), a source

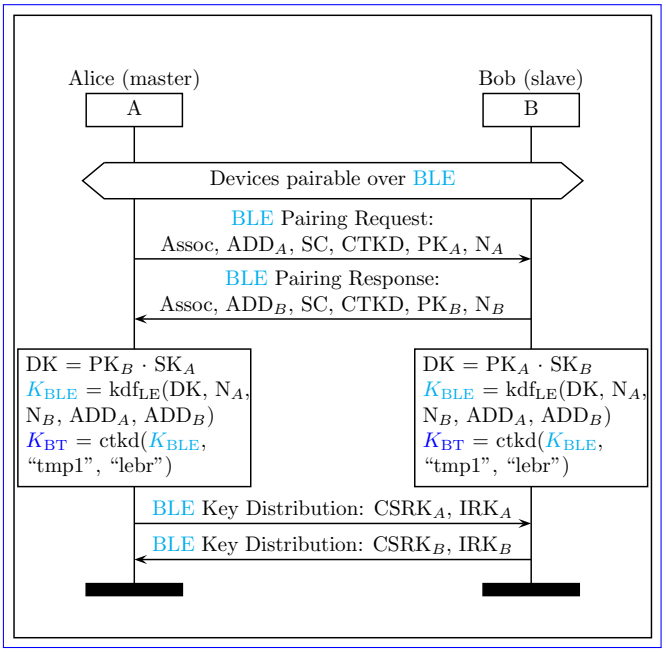

Figure 1: CTKD usage during BLE pairing. Alice and Bob negotiate SC and CTKD support during BLE pairing. Then, they compute the BLE pairing key and from that key, they derive the BT pairing key via CTKD (without exchanging any message over BT). Finally, they generate and exchange additional keys for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

BLE address (ADD), a public key (PK), and a nonce (N). Technically, CTKD support is declared by setting to one the Link Key bits of the Initiator and Responder key distribution SMP fields [? , p. 1680].

After exchanging the pairing messages, Alice and Bob compute a Diffie-Hellman shared secret (DH) using their remote public keys and local private keys (PK). The shared secret is then used to compute the BLE pairing key (K_{BLE}) using a dedicated BLE pairing key derivation function (kdf$_{LE}$). Then, Alice and Bob use CTKD's key derivation function (ctkd) to derive the BT pairing key (K_{BT}) from the BLE key and the static strings "tmp1" and "lebr". Finally, they establish a secure session over BLE and exchange additional keys such as CSRK, and IRK. Once the protocol is concluded, Alice and Bob can establish secure sessions over BT and BLE without having to pair over BT.

CTKD from BT. Figure 2 presents CTKD negotiation during BT pairing. Alice and Bob are pairable over BT and discover each other BT's inquiry mechanisms. 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. Unlike CTKD for BLE, CTKD is *not* negotiated with the BT pairing messages. But, Alice and Bob complete the BT pairing process by computing DH and using it together with their BT addresses and

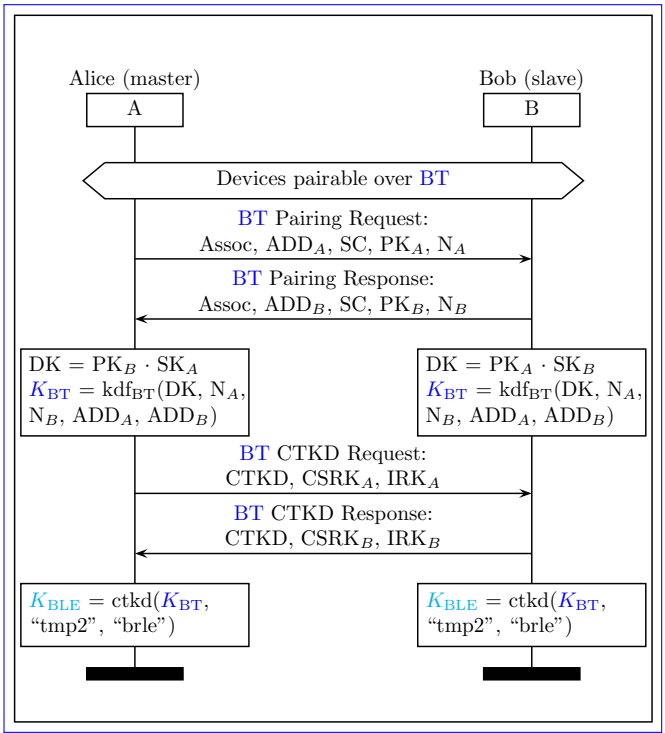

Figure 2: CTKD usage during BT pairing. Alice and Bob during BT pairing negotiate SC support. Then, they compute the BT pairing key, start a secure session over BT and send BT CTKD messages containing CTKD support and other keying material generated for BLE such as signature (CSRK) and identity resolving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

nonces to compute the BT pairing key (K_{BT}) through the BT pairing key derivation function (kdf$_{BT}$).

Then, CTKD negotiation takes place, as Alice and Bob establish a secure BT session and exchange two BT 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 formed by BLE pairing packets (SMP pairing request and response) sent over BT. This is the first example of BLE tunneling over BT that we observed, and the Bluetooth standard so far lacks any diagram or description of this behavior. 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". After the protocol is completed, Alice and Bob can start BT and BLE secure sessions without having to pair over BLE.

CTKD life cycle. By combining all the information acquired from public documents, reverse-engineering implementations, and our

Figure 3: CTKD life cycle has three phases: Discovery (to exchange features), Initialization (to agree on a pairing key and, through CTKD, create a pairing key for the other transport), and Communication (to establish secure sessions on BT and/or BLE).

experiments, we represent the CTKD life cycle for BT and BLE in three phases: Discovery, Initialization, and Communication. Figure 3 shows the life cycle assuming that Alice is a laptop and Bob a pair of headphones. During Discovery, Alice and Bob are pairable on the relevant transport and discover each other. During Initialization, Alice and Bob negotiate SC and CTKD, use one transport (either BT or BLE) to establish a pairing key, and then derive a pairing key for the other transport using CTKD without having to pair a second time. Finally, during Communication the devices are free to establish BT and BLE secure sessions using their shared pairing keys. Each session uses a fresh session key derived from the pairing key and session nonces.

~~To analyse the security of CTKD we introduce our system and attacker models and we describe how CTKD is used in a non adversarial setting. We then introduce the security issues that we discovered with CTKD. These security issues are then exploited by our attacks in Section 4 and addressed with concrete fixes in Section 7.2~~

4 BLUR ATTACKS VIA CTKD

We now present our threat model and the design of four novel and standard-compliant attacks for Bluetooth. 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 the *BLUR attacks*.

4.1 System Model

Our system model considers two victims, Alice and Bob, who ~~want to~~ can securely communicate over BT and BLE. ~~Alice and Bob support CTKDand during pairing and session establishment select the strongest security mechanisms: Secure Simple Pairing (SSP), "Secure Connections", and "Numeric Comparison ". Those security procedures are expected to protect Alice and Bob against impersonation.~~ 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 [**?** , p. 269]. ~~After completing pairing, Alice and Bob can run secure sessions over BT and/or BLE.~~ Without loss of generality, we assume that Alice is the ~~BT and BLE~~ master and Bob is the

BT and BLE slave. Note that we follow the Bluetooth specification of using the terms master/slave instead of more apt terms like leader/follower. 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}. Moreover, we 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 in Bluetooth range with Alice and Bobwho 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 Alice and Bob 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 Alice and Bob and is not present while they pair or establish secure sessionsthe victims, does not have to be present when the victims pair or negotiate a secure session. The attacker can scan and discover BT and BLE devices, jam the Bluetooth channel, pair with Alice and Bob devices, send pairing requests and responses, use CTKD, propose weak association mechanisms (e.g., "Just Works"Just Works), and dissect and craft unencrypted Bluetooth packets.

The attacker has four goals. The first goal one is to impersonate Alice (to Bob) and potentially take over Alice's secure sessions. The second intent 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 attack techniques (i.e., Bluetooth impersonation techniques (i.e., master and slave impersonation attacks). The impersonations).

The attacker's third objective is to establish a man-in-the-middle position in a secure session between Alice and Bob and two victims and requires combining and synchronizing the impersonation attacks on Alice and Bob's impersonation attacks. The fourth goal is to pair and objective is to establish unintended and possibly stealthy sessions with Alice or Bob as an arbitrary device, without breaking existing pairings and secure sessions between Alice and Bob. 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 first demonstrate the CTKD life cycle in a non-adversarial setting to later highlight the CTKD issues (Section 7.1) and attacks (Section 4). The first phase of the CTKD life cycle is *Discovery*, see Figure 3. During Discovery, Alice and now describe our attack strategy using Alice's impersonation as a reference example and with the help of Figure 4. 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 CTKD support. Bobfind each other and exchange their capabilities (e.g., Alice scans while Bob is advertising his presence). During this phase Alice and Bob declare BT, BLE, SSP, and Secure Connections support. Note that the Bluetooth standard does not include CTKD support as a separate feature but it is implicitly activated by declaring BT, BLE, SSP, and Secure Connections.

After Discovery, Alice and Bob initiate *Pairing* that can be performed either over BT or BLE. As a result of pairing Alice and Bob establish a secret pairing key (e.g., or) using SSP with Secure Connections. In particular, this pairing mode uses ECDH to generate a shared secret and a key derivation function that generates the pairing key using as inputs the shared secret, Alice and Bob' ADD, and two nonces. Once Alice and Bobshare a pairing key, then they complete a Bluetooth association phase. There are different association mechanism and in our threat model we assume that Aliceand Bob use a strong mechanism (e.g., Alice and Bob generate the same numeric sequence and the user confirms those).

After association is completed, Alice and Bob run the CTKD key derivation function to compute a second pairing key for the transport that was not used while pairing (e.g., derive from or vice versa). The Bluetooth standard provides a CTKD function that is deterministic, as it uses a pairing key and constant strings (e.g., "br1e" or "1ebr") as inputs [? , p. 1401]. Moreover, even if running a BT session with Alice, has to answer to Charlie with a BLE pairing

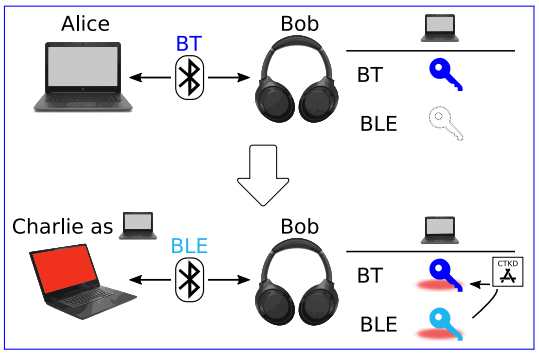

Figure 4: The three phases of the Attack strategy. Alice and Bob are paired over BT and run a secure BT session. Charlie pairs with Bob as Alice over BLE declaring CTKD life cycle: Discovery (to exchange features), Pairing (to agree on support. Then Charlie agrees upon a BLE pairing key with Bob, and, through via CTKD, create a tricks Bob into overwriting Alice's BT pairing keyfor the other transport). As a result, Charlie can establish BT and Communication (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 secure sessions on unintended BT and /or BLE)sessions as an arbitrary device.

response as Charlie's message is compliant with the Bluetooth standarddoes not require to exchange any packet over the air to signal when CTKD is used and the outcome of its usage.

As soon as Aliceand Bob complete Pairing they start the *Communication* phase. During this phase Alice and Bob establish secure sessions over BT and/or BLE. Each session derives a fresh session key from the correspondent Then, Charlie (as Alice) and Bob agree on a BLE pairing key andsession nonces (e.g., from , and from), and uses the session key to encrypt and integrity-protect the link layer traffic with AES-CCM.

CTKDis an interesting attack surface for several reasons. CTKD crosses the security boundary between BT and BLE. Therefore, a CTKD vulnerability is exploitable for both BT and BLE. As CTKD bridges BT and BLE, an attacker can exploit known vulnerabilities on BT to exploit BLE and vice versa. As CTKD is an optional feature and is transparent to the user, an attack exploiting CTKD is hard to detect. As CTKD requires Secure Connections support, an attacker can break the most secure BT and BLE modes by targeting CTKD.

Despite the listed reasons, the Bluetooth standard does not provide a security analysis of CTKD and does not include CTKD in the BT and BLE threat models [? , p. 1401]. As a result, CTKD remains an unexplored attack surface and in this work, we address this concern by performing the first security analysis of CTKD. Our analysis uncovers cross-transport issues (summarized in Table ??). We now describe each issue in detail by using the CTKD life cycle phases presented in Section ??.

Cross-transport issues (CTIs) with CTKD. The issues are at the Bluetooth specification level. SC abbreviates Secure Connections and KO abbreviates Key Overwrite.

CTI 1: Roles (Discovery). During Discovery, Alice and Bob can discover each other and trigger Pairing both over BT and BLE. This is a consequence of CTKD as it enables more ways to pair devices with less user interaction. Alice, as master, is expected to send pairing requests over BT or BLE to Bob, and the user expects to pair Aliceand Bob by discovering Bob on Alice's screen and sending a pairing request to Bob. However, BT master and slave roles are not fixed (unlike BLE) and Alice can receive pairing requests over BT. The attacker can take advantage of this role asymmetry to impersonate a slave device that is already trusted by Alice and send a ***pairing request*** to Alice over BT even if Alice is expecting to receive only BT and BLE ***pairing responses***.

CTI 2: Secure Connections (Discovery). During Discovery, Alice and Bob exchange their capabilities before starting the pairing process. To use CTKD they declare "Secure Connections" support for the transport used for pairing. However, the specification does not specify if CTKD support requires "Secure Connections" support only for the pairing transport or for both transports. From our experiments, we find that CTKD is used when "Secure Connections" is only supported by the pairing transport. This issue considerably increases the CTKD attack surface, as an attacker is not limited to target only devices which support BLE ***and*** BT "Secure Connections" but can also target devices that support BLE ***or*** BT "Secure Connections".

CTI 3: Association (Pairing). During Pairing, Alice and Bob can pair either over BT or BLE. While BT and BLE pairings use different protocols they both include an association phase. The issue is that CTKD does not *enforce the chosen association mechanism across BT and BLE.* This issue can be exploited by the attacker to pair with a weak association mechanism, such as "Just Works", on one transport while the other transport expects a strong association mechanism, such as "Numeric Comparison". This is especially dangerous in case of impersonation attacks because the user is not going to notice an attacker that is re-pairing using "Just Works" pretending to be a trusted device.

CTI 4: Key Overwrite (Pairing). During Pairing, Alice and Bob use CTKD to derive a second pairing term key for the transport not used for pairing. If Alice and Bob already shared a long term key for such transport *CTKD will overwrite the existing pairing key.* This is an issue because an attacker who is impersonating either Alice or Bob can use CTKDto overwrite long term keys. For example, , 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 running a secure session over BT then the attacker can pair with Bob over BLE while impersonating Alice and overwrite the BT key that is shared by Alice and Bobsharing BT *and* BLE pairing keys before the attack takes place.

CTI 5: States (Communication). During Communication, Alice and Bob establish secure sessions over BT and/or BLE. In our experiments, we observed that Alice and Bob *remain pairable over BT and BLE.* Bob also *remains discoverable over BLE.* This is not the case without CTKD where a device is pairable and optionally discoverable only on one transport. This issue gives the attacker more options to discover and pair with victim devices. For example, the attacker can pair on the transport that is not currently in use by Alice and Bob . Furthermore, in some CTKD use cases one transport is supposed to be used only for pairing and deriving keys for the other. Hence, that transport is always in a pairable state but never used after pairing. This enables the attacker to establish unintended malicious sessions on both transports by pairing on the unused one and forcing CTKD.

We now design four novel CTKD cross-transport attacks based on the cross-transport issues that we discuss in Section 7.1. We provide the first attacks that exploit CTKD, blurring the security boundary between BT and BLE. Our attacks are standard-compliant and enable impersonation, interception, and manipulation of traffic between victims, as well as unintended sessions 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 victimdevice. We call our attacks *BLUR attacks*. 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 7.1 for more details). In the remaining

of this section, we describe the technical details of the four BLUR attacks.

4.4 Impersonation Attacks

Master impersonation. Charlie impersonates Alice ~~(master)~~ and takes over her BT ~~secure session~~ and BLE sessions with Bob as in

~~BLUR impersonation attack strategy. Charlie pairs with Bob over one transport (e.g., BLE) and (over)writes the pairing keys for both transports, including Alice's BT pairing key.~~
~~Figure 4 presents the BLUR impersonation attack strategy Before the attack takes place Alice and Bob (the victims) are running a secure BT session and they share a BT long term key (). As a side effect of CTKD, Alice and Bob are pairable on BLE. Charlie (the attacker), targets BLE (which is not used by the victims) and pairs with Bob over BLE as Alice and triggers CTKD, while the real Alice is communicating with Bob over BT. Because of CTKD, Charlie forces Bob to overwrite the BT pairing key that he established with Alice with his own. As a result, Charlie takes over Alice's BT session from BLE. The real Alice can no longer connect to Bob as she does not possess the correct and can attempt to re-pair with Bob only when Charlie terminates his BT session with Bob. Charlie uses the described attack strategy to perform master and slave impersonation attacksas follows:~~

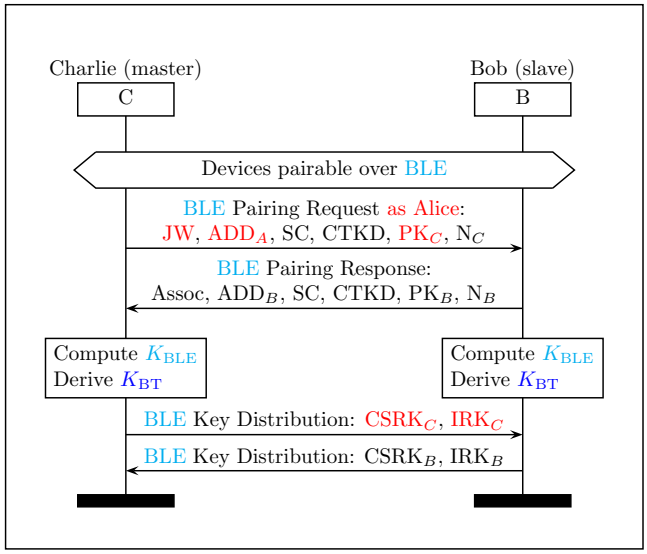

Figure 5: ~~Master~~ BLUR master impersonation attack~~and takeover~~. Charlie sends a BLE pairing request with Alice's address (~~acting as master~~ADD$_A$) ~~pairs with~~ including Just Works (JW) association to avoid user interaction, CTKD, and his public key (PK$_C$). Bob answers with a BLE pairing 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 paring by generating and distributing more ~~key~~keys over a secure BLE session. As a result of the master impersonation attack, Charlie tricks Bob into overwriting Alice's ~~key~~keys with his ones and takes over Alice who can no longer connect back to Bob.

Figure 5. ~~Charlie discovers Bob as he is pairable over BLE and~~ 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$_A$), ~~Secure Connections support (to trigger CTKD), and "Just Works"~~ Just Works (JW) association to avoid user interaction ~~.~~ while pairing, his public key (PK$_C$), and CTKD support.

As Charlie's BLE pairing request ~~does not collide with the BT traffic exchanged by Alice and Bob as BT and BLE use different physical layers and link layers.~~

~~Bob sends Charlie~~ 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 ~~use the exchanged public keys to compute DK. Then they use DK and the exchanged nonces (NC, NB) to compute~~ compute K$_{BLE}$. ~~Then, they locally compute~~ , derive K$_{BT}$ ~~from~~ via CTKD, and exchange additional BLE key material (e.g., ~~using the CTKD's key derivation function (ctkd) . As a result of~~ CSRK, IRK) over a BLE secure session. After the master impersonation attack ~~, Charlie forces Bob to overwrite the BT pairing key that he established with Alicewith his BT pairing key, establishes a BLE pairing key with Bob ,~~ and takes over is completed Charlie takes over Alice's BT and BLE sessions by tricking Bob into overwriting Alice's BT ~~session~~and BLE keys with his ones.

~~BLUR man-in-the-middle attack. The attacker uses the BLUR Impersonation attack against two devices that were previously paired. The two devices do not detect a change but Charlie now has access to all traffic.~~

Slave impersonation. Charlie impersonates Bob ~~(slave)~~ and takes over his BT ~~secure session~~ and BLE sessions with Alice as in Figure 6. ~~In this case Charlie has to wait until the secure BT session between~~ Alice and Bob ~~is interrupted (e.g., by running a master impersonation attack against Bob). Then Charlie can exploit role asymmetries between~~ 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 ~~who is typically expecting pairing responses either over BT or BLE. Charlie's pairing request include Secure Connections support (to trigger CTKD), Bob 's Bluetooth~~ as Bob using Bob's address (ADD$_B$)~~and "Just Works" association to avoid user interaction.~~ , Just Works (JW), and his public key (PK$_C$). Charlie's paring 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, who is pairable over BT, sends a~~ Alice answers with a BT pairing response believing that Bob wants to re-pair over BT~~using CTKD. Charlie and Alice use the exchanged public keys to compute DK. Thenthey use DK and the exchanged nonces to derive (kdf2). Then they locally derive~~ , 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$_{BLE}$ ~~from using CTKD 's~~

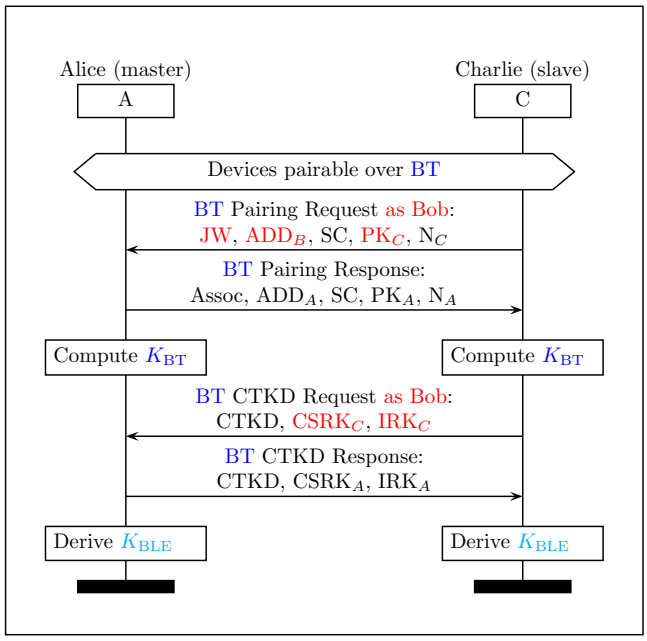

Figure 6: ~~Slave~~ BLUR slave impersonation attack~~and takeover~~. Charlie ~~(acting as slave)~~ sends a BT pairing request with Bob's address (ADD$_B$) including Just Works (JW) association to ~~Alice~~ avoid user interaction, and his public key (~~master~~PK$_C$). The pairing request is valid as 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 ~~key~~keys with his ones and takes over Bob who can no longer connect back to Alice.

~~key derivation functions (etkd). As a result of the slave impersonation attack, Charlie forces Alice to overwrite the BT pairing key that she established with Bob with his BT key, shares a BLE key with Alice, and takes over Bob's BT session. Bob cannot re-establish secure sessions with Alice as he no longer possesses the correct pairing keys.~~

~~As summarized in Table ??, the master impersonation attack takes advantage of all the cross-transport issues that we present in Section 7.1 except CTI 1. In particular, the attacker takes advantage of non-consistent "Secure Connections" support (CTI 2), lack of consistency between~~ via CTKD. Once the slave impersonation attack is completed, Charlie takes over Bob's BT and BLE ~~association methods (CTI 3), more opportunities to pair (CTI 5), and key overwriting (CTI 4). The slave impersonation attack takes advantage of all CTIs except CTI 5, including the role asymmetries between~~ sessions by tricking Alice into overwriting Bob's BT and BLE ~~(CTI 1)~~keys with his ones.

~~Figure 7 presents the high-level description of our BLUR~~

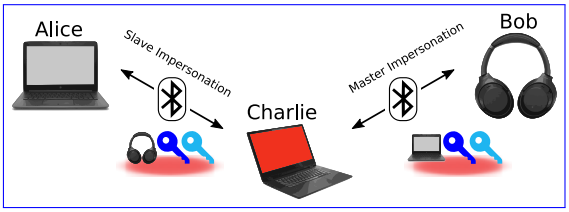

Figure 7: ~~Mapping the requirements of our four~~ BLUR man-in-the-middle attack. Charlie combines the master and slave impersonation attacks presented so far to ~~the discovered cross-transport issues (CTI)~~establish a man-in-the-middle position between Alice and Bob both on BT and BLE.

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 in the previous section,~~ as shown in Figure 7. If Alice and Bob are ~~paired over BT and they run a secure sessionover BT. During this attack, Charlie sequentially performs the master and slave impersonation attacks described in Section 4.4. As a result, the attacker overwrites Alice and Bob's BTpairing keys with known keys, establishes BLE long term keys with Alice and Bob~~ running a BLE session, ~~and positions himself in the middle to access all traffic between the victims and to inject valid traffic both on~~ 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

~~Figure ?? shows the details of the MitM attack. Firstly, Charlie impersonates Alice to Bob over BLE (as in Figure 5), overwrites Bob's BT key with his key (K$_{BC}$). Secondly, Charlie impersonates~~

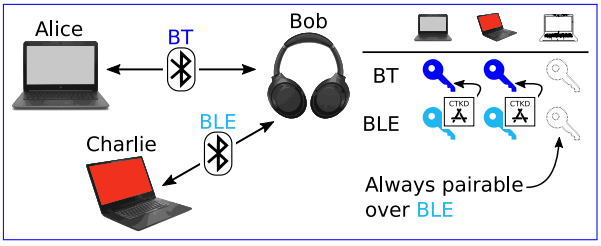

Figure 8: ~~MitM~~ BLUR unintended sessions attack~~and takeover~~. Charlie ~~impersonates Alice as in Figure 5, impersonates~~ can take advantage of CTKD to establish unintended BT and BLE session with Bob as ~~in Figure 6, let the victims mutually authenticate and then gets access to their traffic~~a random device with arbitrary capabilities. The same can happen if Charlie targets Alice.

Bob to Alice over BT as in Figure 6 and overwrites Alice's BT key with his key (K_{AC}). Then, Alice and Bob exchange two nonces (N_A, N_B) to authenticate the BT pairing key. Charlie mutually authenticates with Bob and Alice by using a message authentication code (MAC)function keyed with the appropriate key and input parameters. Finally, So far we described how to exploit CTKD to impersonate any Bluetooth device, however, the attacker can also take advantage of CTKD to establish unintended BT and BLE sessions with a victim as an anonymous device with arbitrary capabilities. Unintended sessions are interesting because they expose a larger attack surface than a setup where the attacker can only send scanning or advertising packets to a victim (i.e., when the victim does not trust the attacker). For example, by establishing unintended sessions, the attacker can enumerate all BT and BLE services supported by the victim and exploit a remote code execution vulnerability that would not have been exploitable without a secure session. Concurrently, these attacks are more difficult to spot than impersonation ones as they do not require to take over existing secure bonds (i.e., they do not require to overwrite keys).

Let us see how an unintended session attack works in a scenario where Alice and Bob establish are already paired and are running a secure BT session with Charlie in the middle, and Charlie gets access to all traffic exchanged by Alice and Bob and can modify and inject arbitrary valid traffic between Alice and Bob.

As summarized in Table ??, the BLUR man-in-the-middle attack is a composition of the master and slave impersonation BLUR attacks and takes advantage of all the CTI that we present in Section 7.1.

BLUR unintended sessions attack. Charlie sends a BLE pairing request to Bob (who remains pairable over BLE due to CTKD) as an unknown device with arbitrary capabilities. After CTKD completes, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking Bob's existing pairings and sessions.

Figure 8 presents a BLUR unintended session attack targeting Bob. In this (see Figure 8). As in the impersonation attack scenario, Alice and Bob are running a secure session over BT but they are still must also be pairable over BLE in order to accept pairing requests with other devices and run to support CTKD. Charlie targets Bob (slave) by sending him a pairing request over BLE as an unknown device. Charlie can pretend to be any device having arbitrary capabilities, e.g., by sending a BLE pairing request using a random Bluetooth address, Bluetooth name, device class, "Secure Connections" CTKD support, and weak Just Works for association. Bob , accepts to pair with Charliewhile continuing his session with Alice. Then, Charlie and Bob answers to Charlie's request and the two negotiate K_{BLE}, and derive K_{BT} using via CTKD. Now, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking his existing pairings or sessions with other devices Bob's existing sessions (e.g., with Alice) .

Charlie can also establish unintended sessions with Alice (master). In particular, he can impersonate a BLE slave and start advertising his presence. Once Alice discovers Charlie, she can establish a BLE connection with him, and Charlie can explicitly request to pair using a SMP Security Request packet [? , p. 1401]. Then, Alice and Charlie compute , and derive using CTKD. Nowand by using an anonymous identity and arbitrary capabilities. Using a similar strategy, Charlie can establish secure but unintended BT and BLE sessions with Alice without breaking her existing pairings or sessions with other devices (e.g., with Bob). Charlie can take advantage of the unintended sessions with Aliceand Bob in many ways. For example, he can use the session to drop known exploits such as BlueBorne [?] , BLEEDINGBIT [?], or SweynTooth [?], new exploits, and to enumerate and tamper with BT and BLE services and characteristics (including the protected ones). reach the same goals targeting Alice.

Those attacks are particularly effective when the victims are using one transport only to pair and derive keys with CTKD. For example, a Bluetooth speaker only streams music over BT but is also pairable over BLE to enable users to discover it without having to put it into BT pairing mode. As summarized in Table ?? the unintended session BLUR attack takes advantage of CTI 2 and CTI 4.

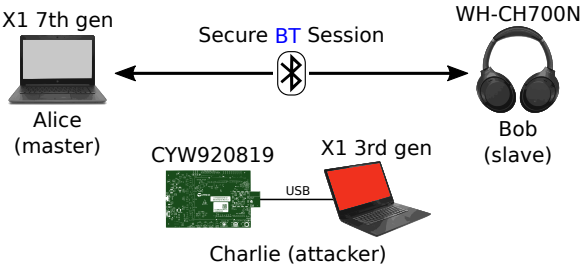

Figure 9: Example BLUR Attack Scenario. Alice (master) is a ThinkPad X1 7th gen, Bob (slave) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in absence of Charlie, and are running a secure BT session.

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. The tools that we developed will be open-sourcedWe will fully open-source both the attack and our CTKD key derivation functionality.

5.1 Attack Scenario

Our attack scenario follows the example in Figure 9 and includes two victims, Alice (master) and Bob (slave). In Figure 9 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 [?] 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 with different to attack other victim devices.

To understand the capabilities of the victims and the attacker we summarize their most important Bluetooth features in Table 1. We note that Bobis capable of using CTKD over BLE even if he does not support "Secure Connections" over BT and does not support Bluetooth version 4.2. This confirms the "Secure Connections " cross-transport issue (CTI 2) that we discuss. Furthermore to conduct the attacks we had to develop an attack device that enabled us to

	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 1: Relevant Bluetooth features for Alice, Bob, and Charlie~~in our example attack scenario~~. ~~Alice and Bob support CTKD even if Bob's Host does not support BT SC (BT "Secure Connections").~~ We redact the devices' Bluetooth addresses for privacy reasons.

~~change all the features in Table 1. Some of those features, such as the version and subversion numbers, are particularly challenging to modify as they require patching a Bluetooth firmware that is typically proprietary and closed-source~~Table 1 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

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

~~To implement the BLUR attack we had to develop~~ To conduct our attacks we developed a custom attack device ~~. As we can see from its block diagram in~~ making use of a CYW920819 development board connected to a Linux laptop (see Figure 10~~, the attack deviceconsists of a Linux laptop implementing the Bluetooth host component using BlueZ (i.e.,~~). 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

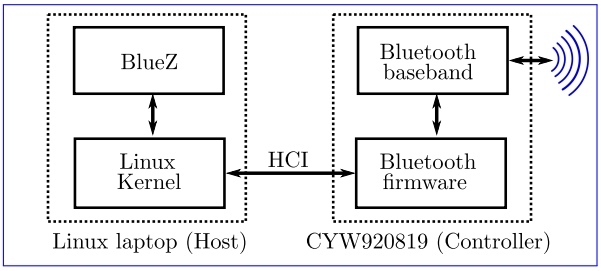

Figure 10: Attack Device Block Diagram. The attack device is composed of Linux laptop (Host) and a CYW920819 development board (Controller) connected via USB and communicating using the Host Controller Interface (HCI) protocol.

(e.g., ~~user-space) and the Linux kernel. The laptop is connected via USB to a CYW920819 development board . The board implements the Bluetooth controller using a firmware and a baseband. The laptop and the board support BT , BLE , SSP, Secure Connections, and CTKD and they communicate using the Host Controller Interface (HCI) protocol over USB~~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 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 ~~used~~ 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 ~~, unlike other tools, it enables to decide the type of pairing request and declared association mechanism.~~ 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 ~~wanted to access the traffic exchanged over the air by our attack device~~configured our host to get all link-layer packets sent and received by the controller. This is ~~not available on a standard Bluetooth device. To achieve this goal~~ 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 ~~enable diagnostic mode on~~ the controller that switches on an undocumented diagnostic mode in the controller. ~~This mode tells the board to copy all the BT and BLE link-layer packets and send them over HCI to the host.~~ Then, we added extra C code to the Linux kernel to parse those ~~HCI packets. With this setup, we can monitor both~~

HCI and link-layer traffic directly from the hostwithout requiring over-the-air BT and BLE sniffersspecial HCI packets in the host.

Modifying the controller required us to interact directly with

Controller modifications. We modified the controller by dynamically patching the development board 's Bluetooth firmware . To extract the 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 from Cypress to read and save a runtime RAM snapshot from the board's SoC. We took the snapshot after the firmware was initialized to acquire the firmware patches applied at runtime. We use the memory maps that we extracted from the board's SDK to extract the various memory segments from the snapshot including the ROM, the (e.g., ROM, RAM, and the scratchpadsegments). 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 segments in Ghidra (a free and open-source decompiler and disassembler)in Ghidra and statically analyzed them. In our first reverse-engineering passpass, 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 the debugging these symbols we isolated functions and data structures relevant for to the BLUR attacks. Then, we wrote ARM Thumb assembly patches to change their behaviors and we apply those patches at runtime using internalblue [?], an open-source toolkit to manage several Bluetooth devices including our board. Our set of patches allow modifying crucial capabilities and parameters declared by the controller including the Bluetooth address and name, device class, Secure Connections support, and authentication requirements (as shown in Table 1). allows transforming our board in whatever device we want by changing its identifiers including addresses, names, and capabilities.

5.3 CTKD Key Derivation Function

Our BLUR attacks leverage CTKD, so the first step of our evaluation requires to confirm that the devices under test support and (correctly) implement it . As CTKD is an optional feature and it is not negotiated with a dedicated flag, we can only speculate that a device supports it if it declares Secure Connections support for BT and BLE. Furthermore, there are no available tools to check the correctness of the keys derived via CTKD.

To address those issues we implemented the CTKD derivation function based on the Bluetooth standard [? , p. 1401]. Our implementation We implemented CTKD's key derivation function, following its specification in the Bluetooth standard [? , 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 [?], was successfully tested against the standard's test vectors and the CTKD keys produced during our attacks. To enable other researchers to investigate CTKD we will open-source our implementation. We tested it against the CTKD test vectors in the standard [? , p. 1721]. We now describe its technical details.

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

CTKD function for BT (top) and BLE (bottom). The functions are the same but use a sequence of two AES-CMAC with different input quantities. In the first AES-CMAC, the devices use a constant string as key and the pairing key as input if they support the h7 conversion function, otherwise, they swap the two. In the second AES-CMAC, the devices use the MAC from the first stage as key and a constant string as the input to derive the cross-transport pairing key.

We now describe the CTKDkey derivation function implementation details. The Bluetooth standard specifies a single CTKD function (see Section 3.1) that is used with different parameters for BT and BLE. Figure ?? shows the CTKD key derivation function for BT (top) and BLE (bottom). Both use a chain of two AES-CMAC blocks in sequence with different keys and 4-byte constant strings. AES-CMAC is a message authentication code (MAC) based on the AES block cipher [?]. In particular, BT uses , "tmp2" and "br1e" and derives We implemented CTKD's key derivation for BT deriving and following the equation above. The key derivation computes K_{BLE}, while BLE uses , "tmp1" and "1ebr" and derives .

In the first using a function $f(a, b)$ that corresponds to AES-CMAC, if both devices support the $(key, plaintext)$. 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 conversion function is a key conversion function defined in the Bluetooth standard [? , p. 1634], the long term key is used as key and the string as input, otherwise, the string (padded with 12 zeros) is used as key and the long term key as input and is negotiated during pairing using AuthReq [? , p. 1634]. In the second AES-CMAC, the 128-bit (16-byte) output of the first AES-CMAC is used as key and the string as input. The 128-bit (16-byte) output of the second AES-CMAC is the derived long term key.

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

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 conducted the BLUR attacks and our evaluation results on 13 unique devices devices using 10 unique Bluetooth chips (see Table 2). The tested devices represent popular laptops, phonesOur evaluation exploit different device types (e.g., laptops, smartphones, headphones, and an embededd platform. The devices are from a broad set of device producers (development boards), manufacturers (e.g., Samsung, Dell, Google, Lenovo, and

Sony), ~~run different~~ operating systems (e.g., Android, Windows, Linux, and proprietary OSes), ~~use different Bluetooth chipsets (from~~ and Bluetooth chip (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 ~~unintentional~~ unintended session attacks. In the next paragraphs, we describe how we conducted them using our custom attack device described in Section 5.2.

Laptop (master) BLUR impersonation attack. To impersonate the laptop, we ~~configure~~ patch our attack device to clone the laptop~~Bluetooth features,~~ 's Bluetooth features (including Bluetooth address, Bluetooth name, device class, ~~BT and BLE "Secure Connections" support, and advertised services. We accomplish this task by patching~~ SC, and CTKD support). Then, we send a BLE pairing request from the attack device ~~'s Bluetooth firmware and configuring the attack laptop accordingly. Once the attack device looks like the impersonated laptop, we ask the headphones to pair over BLE using "Just Works " and CTKD.~~ 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 ~~request to pair over BLE, update the BLE long term key, run CTKD for BT, update the BT long term key,~~ pairing request, and the devices agree on K_{BLE}, derive K_{BT} via CTKD and establish a secure BLE session~~with the attack device~~. 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 ~~new BT and BLE long term keys~~ correct pairing keys overwritten by the attacker.

Headphones (slave) BLUR impersonation attack. To impersonate the headphones, we ~~configure~~ patch our attack device to clone the headphones~~Bluetooth features using the same technique adopted for the laptop impersonation. Once~~ ' Bluetooth features. Then, we send a BT pairing request from the attack device ~~looks like the impersonated headphones we ask the laptop to pair over BT using "Just Works " and CTKD. The malicious BT pairing request is sent~~ to the laptop declaring CTKD and Just Works support using `btmgmt`'s TUI. The laptop accepts to pair over BT, ~~updates the BT long term key, and runs CTKDfor BLE. Then, we~~ and the devices agree on K_{BT}, negotiate CTKD, derive K_{BLE} via CTKD, and establish a secure ~~BT session with the headphones~~ session over BT. The impersonated headphones cannot connect to the laptop as they do not own the correct paring keys.

To ~~evaluate~~ optimize the evaluation of the master and slave impersonation ~~attack experimentally~~attacks, we used the attack device both as the attacker and ~~as one of the victims~~the impersonated victim. For example, in a master impersonation attack we pair the attack device with the slave victim device, we disconnect them, we "forget" the victim device on the attack device and we run the master impersonation attack from the attack device. This setup is efficient ~~,~~because it allows us to quickly test many slave victims. For the slave impersonation, we use the same procedure ~~and quickly test many~~ to test our master victims.

BLUR Man-in-the-middle attack. By using our BLUR implementation with two development boards connected to the same attack 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.

~~If a victim device is vulnerable to the master or slave impersonation attack, then is also vulnerable to the man-in-the-middle attack , as the latter requires a vulnerable master device and a vulnerable slave device .~~

~~*Unintended sessions attack.* To perform the unintended sessions attacks, we configure the attack device to impersonate an arbitrary device with arbitrary services over BT and BLE. Then we send a malicious pairing request to the headphones over BLE and one to the laptop over BT. Both pairing requests declare support for CTKD and "Just Works". The attackdevice establishes new BT and BLE keys both with the headphones and the laptop and starts unintended sessions with both~~

BLUR Unintended sessions attack. For the unintended session attack, we patched our attack device to look like an unknown device to the current victim. 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 creates unwanted but trusted bonds with a victim and can establish secure sessions over BT and BLE ~~.~~

~~We test this attack by connecting the target victim to a third device and then by trying to establish unintended sessions~~ with the victim~~as an arbitrary device over the transport that is not used by the legitimate connection. For example, if the victim is a pair of headphones that is connected with a laptop over BT then we run the unintended session attacker over BLE. .~~

6.2 Evaluation Results

~~BLUR attacks evaluation results. The last three columns contain a checkmark () if a device is vulnerable to the master impersonation attack (MI), slave impersonation attack (SI), man-in-the-middle attack (MitM), or unintended session (US) attack. If the victim's role is slave then we test the victim against a master impersonation attack (Role = Master), otherwise, we test it against a slave impersonation attack (Role = Slave), and we group the attacks in one column (MI/SI column). As shown by the last three columns, all the tested devices (unique Bluetooth chips) are vulnerable to all relevant BLUR attacks.~~

We evaluated the BLUR attacks ~~on~~ against 13 ~~devices, the~~ unique devices (employing 10 unique Bluetooth chips) and our results are summarized in Table 2. The first six columns indicate the device~~producer~~'s producer, model name, operating system, ~~device model , OS,~~ chip manufacturer, chip model, and ~~supported~~ Bluetooth version. The seventh column ~~indicates the attacker role. The~~ contains either Slave if the device was tested against a slave impersonation attack, or Master if the device was tested against a master impersonation attack. The table's last three columns contain a ~~checkmark~~ check

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	✓	✓	✓
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	✓	✓	✓
Sony	WH-1000XM3	Proprietary	CSR	12414	4.2	Master	✓	✓	✓
Sony	WH-CH700N	Proprietary	CSR	12942	4.1†	Master	✓	✓	✓

† CTKD functionality was backported by the vendor to Bluetooth 4.1 for this device.

Table 2: 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 produce and model. The sixth column tells the Bluetooth version of the target device. The seventh column indicates the attacker's role (e.g., if Slave then the attacker is the slave and targets a master). Finally, the last three columns contain a check mark (✓) if a device is vulnerable to the relevant BLUR attack. We group master and slave impersonation attacks in the same column (MI/SI) as each victim can only have one role. All the devices that we tested are vulnerable.

mark (✓) if a device is vulnerable to ~~the master impersonation attack (MI),~~ master or slave impersonation attack (MI/SI), man-in-the-middle attack (MitM), or unintended session (US) attack. ~~The master and slave impersonation attacks are grouped in one column (MI/SI column). If the victim's role is slave then we test it against a master impersonation attack, otherwise, we test it against a slave impersonation attack. As shown by the last three columns, all the devices (unique Bluetooth chips)~~

From Table 2 we can draw several significant conclusions. Firstly, it shows that the BLUR attacks are practical as all devices that we tested are vulnerable~~to all relevant BLUR attacks.~~

~~As we tested a wide range of devices that were all vulnerable, our evaluation demonstrates that~~ . Secondly, the table demonstrates that all the Bluetooth versions that we tested are vulnerable, i.e., Bluetooth versions 4.1, 4.2, 5.0, and 5.1. Finally, the table confirms that the ~~BLUR attacks are practical, standard-compliant, and affect all the Bluetooth versions that support CTKD. As the~~ BLUR attacks are standard-compliant ~~, potentially all standard-compliant devices supporting CTKD are also vulnerable. Based on our evaluation, we suggest that the Bluetooth SIG fix the issues that we uncover in CTKD and we provide our set of countermeasures for the Bluetooth standard in Section 7.2~~as they work regardless of device-specific implementation details.

7 DISCUSSION

~~We now discuss the lessons learned and our set of countermeasures to mitigate the~~ In this section, we describe four issues affecting CTKD's specification that we extrapolated from the BLUR attacks.

~~There are several lessons that we learned while analyzing CTKD and developing the~~ These issues are standard-compliant and represent the root causes of the BLUR attacks. ~~In this section we report those lessons evaluating~~ Furthermore, we propose four effective countermeasures to address the BLUR attacks. ~~In this section we report those lessons as they are useful for protocol designers who are dealing with cross-transport features and related security issues~~Finally, we discuss the main lessons learned.

~~*Cross-transport mechanisms need a cross-transport threat model.* Security mechanisms, such as CTKD, that cross the security boundary between two technologies with different threat models should be designed using a~~

7.1 Cross-Transport Issues with CTKD

We isolated four cross-transport ~~threat model. For example, the Bluetooth standard should consider that an attacker might try to exploit BT from BLE via CTKDand vice versa. Unfortunately, at the time of writing, the Bluetooth standard lacks a cross-transport threat model. The lack of a threat model (along with a security analysis) is the main reason why we were able to uncover severe issues with CTKD~~ issues (CTI) with the specification of CTKD.

~~*Similar security mechanisms with different threat models do not provide the same security guarantees*~~. All CTI are related to the cross-transport nature of CTKD, i.e., the fact that CTKD bridges BT and BLE ~~both provide their version of pairing and secure session establishment. One might think that pairing over BT and then establishing a secure session over BLE provides the same security guarantees of~~

pairing over BT and establishing a secure session over BLE. However, this is not the case, as those mechanismsare similar but not equal and they are designed with different threat models in mind. Mixing those procedures actually enables more ways to attack BT and BLE (e.g.,security domains without properly enforcing certain aspects. We now describe each of them in detail.

CTI 1: Role Asymmetry. BT and BLE define their master and slave roles differently. In particular, BT enables to switch roles dynamically on demand, while for BLE the roles are fixed. The attacker takes advantage of this CTI by acting as a master for BT and a slave for BLE. For example, in the slave impersonation attack, the attacker can send a BT pairing request, when the victim would expect to receive only BT responses.

CTI 2: Association Asymmetry. The Bluetooth standard does not mandate to enforce the same association mechanism for BT and BLE. An attacker can take advantage of this issue to use a weak association method on one transport when the other is expecting to use a stronger association mechanisms. For example, in the master impersonation attack even if the victims have paired over BT with strong association, the attacker can pair over BLE with weak association (i.e., the BLUR attacks) Just Work) and impersonate a device.

Properly weighting usability against security benefits is keyCTKD was introduced to improve CTI 3: Key (Over)write. With CTKD the Bluetooth standard introduces a new attack primitive, that is, cross-transport key (over)writing. The attacker can use such primitive to overwrite trusted pairing keys and distribute new BT and BLE usability. In light of the presented issues and attacks , we learned that the usability benefits introduced with CTKD are not balancing the security issues introduced by CTKD. We agree that no-one wants to use complicated security mechanisms, but the Bluetooth standard should have introduced a secure and usable CTKD mechanismkeys. All the presented attacks take advantage of this issue.

CTI 4: Pairing States. With CTKD the Bluetooth standard enables more ways to pair devices. The attacker can take advantage of this issue to target the transport not currently in use by the victims. For example, in the master impersonation attack, the attacker sends a pairing request over BLE while the victims are using BT.

7.2 Countermeasures

We now present a set of countermeasures to address all the five cross-transport issues (CTI) that we present four countermeasures to mitigate the BLUR attacks presented in Section 7.1. Our countermeasures 4. In particular, the first three mitigations defeat the BLUR impersonation and MitM-attacks, while the unintended session attacks are prevented by deploying the fourth mitigation. The countermeasures are also addressing the CTI issues described in Section 7.1, and can be implemented in on the device's Bluetooth Host (i.e., device's OS) , OS) by storing and checking extra metadata about its state and trusted remote devices. list of trusted devices. We argue that the Bluetooth Host is the natural place to store this new metadata in addition to other metadata such as long term keys.

Align BT and BLE roles (CTI 1). The BLUR attacks take advantage of BT and BLE role asymmetries to act as a BT master while being a BLE slave. To fix this issue

Align BT and BLE roles. To fix role asymmetries between BT and BLE, a device should store the role that the remote device used while pairing and enforce it across re-pairings. In case of a role mismatch, the device should abort pairing.

Enforce Secure Connections (CTI 2)strong association mechanisms. In our experiments, we can use CTKD with the WH-CH700N headphones even if they only support "Secure Connections" for BLE. This should not happen as CTKD should be used only when "Secure Connections" is provided by both To align association methods between BT and BLE and pairings, a device should enforce this condition before running CKTD and abort CTKD if this condition is not met.

Enforce strong association mechanisms (CTI 3). BT and BLE do not protect the negotiation of the association mechanism and CTKD allows two devices to use different association mechanisms on different transports when pairing and re-pairing. The BLUR attack exploits this fact to re-pair with a victim device using "Just Works" even if the victim supports "Numeric Comparison". A device should keep track of the remotes' strongest association mechanism used while pairing either on BT or BLE and enforce it for subsequent (re-)pairings. If a weaker mechanism than the one stored is proposed, pairing should be aborted.

Disable CTKD key overwrites(CTI 4). CTKD allows (over)writing BT long term keys from BLE and vice versa. This enables an attacker impersonate a device and take over her existing session on one transport by attacking the other. To fix this issuesecurity keys across BT and BLE. To fix key overwrites via CTKD, a device should disallow key overwrites with to update a trusted key via CTKD when a paired device wants to re-pair. For example, re-pairing over BT should not overwrite a BLE long term pairing key that was securely established in the past. When a device has lost a long-term key for a transport (e.g., device reset), it should explicitly re-pair on that transport.

Disable pairable state pairing when not needed(CTI 5). In our experiment we confirmed that a device might remain pairable over BT and BLE even after it has paired and is communicating with a remote device . This is problematic as an attacker can target the transport that is not currently used by the two devices to launch the BLUR attacks. To address this issueTo prevent an attacker from pairing with a victim device in unexpected ways, a device should automatically stop being pairable on a transport that is not currently in use. For example, a pair of headphones who are running a secure session over BT with a laptop should not answer pairing requests over BLE unless the user explicitly renters set the headphones in pairing mode.

7.3 Lessons Learned

There are several lessons that we learned while analyzing CTKD and developing the BLUR attacks. We report them in the hope that they will be useful for protocol designers who are dealing with cross-transport features and related security issues.

Cross-transport mechanisms need a cross-transport threat model. Security mechanisms that cross the security boundary between two technologies should be designed and tested against a cross-transport threat model. For example, the Bluetooth standard should include

			Attack				
Year	Paper	Target	Phase	C I A K	SC/SCO	Persistent	Note
Attacks on BT							
2016	Albazrqaoe et al. [?]	Standard	Any	◐○○○	x	-	BlueEar Sniffer
2017	Seri et al. [?]	Impl.	Pairing	●●●○	NA	✓	BlueBorne
2018	Sun et al. [?]	Standard	Pairing	●●●○	✓	-	Passkey (MitM)
2018	Biham et al. [?]	Impl.	Pairing	●●●◐	NA	✓	Fixed Coordinate Invalid Curve
2019	Antonioli et al. [?]	Standard	Pairing	●●◐○	✓	-	KNOB (MitM)
2020	Antonioli et al. [?]	Standard	Pairing	●●●○	✓	-	BIAS
2021	Tschirschnitz et al. [?]	Standard	Pairing	●●●○	✓	-	Method Confusion (MitM)
Attacks on BLE							
2016	Jasek et al. [?]	Standard	NA	◐○○○	x	-	Black Hat
2019	Seri et al. [?]	Impl.	NA	○◐◐○	NA	✓	Bleedingbit
2020	Zhang et al. [?]	Standard	Pairing	◐◐◐○	✓	-	MitM (SCO)
2020	Wu et al. [?]	Standard	Session	○○●○	✓	-	BLESA
2020	Garbelini et al. [?]	Impl.	Any	◐◐◐○	NA	-	SweynTooth fuzzer
Attacks on both BLE and BT							
2019	Ossmann et al. [?]	Standard	NA	◐○○○	x	-	Ubertooth sniffer
2020	Antonioli et al. [?]	Standard	Pairing	●●◐○	✓	-	Downgrade (MitM)
2021	**This work**	Standard	Any	●●●◐	✓	✓	

Table 3: Overview of recent attacks on Bluetooth and BLE. C = Data Confidentiality, I = Data Integrity, A = Device Authentication, K = Key disclosure. No (○) Partially (◐), Yes (●).

in its threat model an attacker who wants to exploit BT from BLE and vice versa, rather than considering only attackers focused either on BT or BLE.

Security mechanisms used to cross a security boundary should provide the same security guarantees. Cross-transport mechanisms should be designed such that the mechanisms used to cross the security boundary provides the same security guarantees in the same threat model. Currently, this is not the case for Bluetooth as CTKD uses BT and BLE pairings to cross the security boundary and pairing over BT is different than pairing over BLE.

Usability should not outweigh 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 also introduced with CTKD. Indeed, it is paramount to weight security and usability before introducing a critical security feature, especially if it allows crossing a security boundary.

8 RELATED WORK

We summarize the positioning of our attacks compared to to related work in Table 3, and provide additional details on those attacks in the following. In general, the BLUR attacks are the first cross-transport attacks (targeting both BT and BLE), are standard-compliant (i.e., expected to work on any device that supports CTKD), can be executed outside the victims' initial pairing phase, provide comprehensive compromise of the security properties, break the most secure Bluetooth modes (secure connections), and provide persistent compromise of the victims.

The Bluetooth provides a royalty-free and widely-available cable replacement technology [?]. 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 [? ?]. Standard-compliant attacks on BT include attacks on legacy pairing [?], secure simple pairing (SSP) [? ? ?], Bluetooth association [? ?], key negotiation [?], and authentication procedures [? ? ?]. Standard-compliant attacks on BLE include attacks on legacy pairing [?], key negotiation [?], SSP [? ?], reconnections [?], and GATT [?]. 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 [?] and BLE [? ?]. 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 [? ? ?] 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 [?] and Google Nearby Connections [?]. 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 [?] and it is considered relatively weak [?]. SAFER+, used for authentication, was analyzed as well [?]. BT and BLE "Secure Connections" use the AES-CCM authenticated-encryption cipher. AES-CCM was extensively analyzed [? ?] and it is ~~FIPS~~

compliantFIPS-compliant. Our BLUR attacks target key negotiation and not cryptographic primitives, and are effective even with perfectly secure cryptographic primitives.

9 CONCLUSION

We present the first security analysis of CTKD and identify novel standard-compliant and 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 issues and attacks against attacks on Bluetooth, they are standard-compliant, and effective against the most secure BT and BLE . Our attacks show that CTKD enables an attacker to cross the security boundary between 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 previously published attacks on the individual BT and BLE transports, our attacks on CTKD do not require the attacker to be present during pairing or secure session establishment. As a result, our attacks have lower requirements for the attacker while still allowing to break BT and BLE security guarantees. 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.

We identify cross-transport issues related to roles (CTI 1), "Secure Connections" (CTI 2), association (CTI 3), key overwrite (CTI 4), and pairing states (CTI 5). Based on those issues, we develop attacks against BT and BLE enabling impersonations, traffic manipulation, and malicious session establishment. We name our attacks 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.

We provide and discuss a low-cost implementation To demonstrate the practicality of the BLUR attacksusing off-the-shelf hardware and , we presented a low-cost implementation based on cheap readily available hardware (a laptop, and a Bluetooth development board) and open-source software. To demonstrate that our attacks are practical, we successfully exploit devices from different hardware and software manufacturers. Our devices range across all the Bluetooth versions supporting CTKD (e.g., versions greater or equal to 4.2) and also a version of Bluetooth 4.1 with backported CTKD features.

We discuss several lessons that we learned (e.g., the importance of a cross-transport threat model) and the major technical challenges that we faced (e.g., 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 present five countermeasures to mitigate the BLUR attacks. Each countermeasure addresses a specific cross-transport with a concrete fix that can be implemented at the Bluetooth standard level. We responsibly disclosed our vulnerabilities, attacks, and countermeasures to the Bluetooth SIG.

We use our implementation to experimentally confirm that CTKD-compatible devices (using 10 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.

=10000