Response to MR (paper #129)

In this document we summarize the changes that we made to address the reviewers's comments for a Major Revision. In particular, in each section we address a specific comment and tag the relevant reviewer(s). Together with our MR response document we attach the paper diff.

We thank the reviewers for the opportunity to provide a major revision of our paper. We have carefully analyzed the reviews and addressed all open issues to the best of our understanding. We remain open to discuss any further required changes.

Provide more technical detail on the protocols (Rev. A,B)

We improved Section 3 to provide more technical information about the attacked protocols. This information enables the reader to verify the attacks presented in Section 4 and to assess the five cross-transport security problems with CTKD that we discuss.

In particular, we introduced a new section (Section 3.3) to describe CTKD and the related Bluetooth protocols in non-adversarial setting with the help of Figure 2. Moreover, we improved Section 3.4 to better explain what are the identified issues and why those issues are not overlapping between each other. We also summarize the salient information about the issues in a dedicated table (Table 1).

Provide more technical detail on the reverse engineering (Rev. D)

We improved Section 5.2 to provide more technical information. In particular, we explain how we modified the host components (e.g., customized Linux kernel) and the controller components (e.g., reverse-engineering and runtime modification of a Bluetooth firmware) and introduced a new figure (Figure 10) to introduce our custom attack device. All components and modifications will be released as open-source with the publication of the paper.

Add an in-depth analysis of the causes and fundamental problems (Rev. B)

We improved Section 3 to better present and analyze the fundamental problems with CTKD (i.e., the five CTI vulnerabilities). In Section 7.1 we explain what we highlight the causes of such problems by answering questions such as "Why were those vulnerabilities there?" and "How should we design cross-transport mechanisms to avoid such vulnerabilities?"

Develop more insights on sound defense strategies (Rev. A,B,C,D)

We improved Section 7.2 by providing five countermeasures where each one addresses a specific CTI with a concrete fix. We provided similar countermeasures when we disclosed the issues to the Bluetooth SIG, but the SIG has not released any official patch or mitigation yet.

Include responses and status from the responsible disclosure with the Bluetooth SIG (Rev. D)

We updated the last paragraph of the Introduction with the latest information about the public disclosure articles about the BLUR attacks. In particular, we disclosed our findings and countermeasures to the Bluetooth SIG in May 2020. The Bluetooth SIG acknowledged our findings and assigned CVE-2020-15802 to the BLUR attacks. In September 2020, the Bluetooth SIG released a security note about our report (without contacting us).

Position the identified issues properly wrt. implementation vs. specification issues

The presented issues are at the Bluetooth specification level and the Bluetooth specification does not provide implementation guidelines to address them. While all the devices that we tested are affected by the presented issues, an implementer can mitigate some issues in a standard-compliant way. For example, a device might stop being pairable on a transport when is not in use. To better communicate this message we updated the abstract, the introduction, Section 3, and the conclusion.

Put the attacks in the context of prior work, including attacks on BT/BLE as well as other protocols, convince the reader that your literature search was exhaustive, and clarify which steps of the attacks are novel (Rev. B)

In the Introduction we revised the presentation of the attacks in the context of prior work, in particular regarding requirements to conduct the attack. Unlike prior work, our attacks do not require the attacker to be present during pairing or secure session establishment. Therefore, our attacks enable Bluetooth exploitation in new scenarios, in which attacks were not possible before.

Tone down claims and overselling as indicated in the reviews (Rev. A)

We revised our tone and positioning to more precisely communicate our contribution. In particular, we rewrote our section headers, and several statements in

the Introduction, Section 3, and the Conclusion with a more appropriate tone and removed redundant claims.

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

Anonymous Submission #129

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 have different security architectures and threat models~~, but they use similar security mechanisms~~. In particular, pairing enables two devices to establish a long term key to secure the communication. Two devices have to pair over BT and BLE to use both transports securely. Since pairing the same devices ~~two times~~ twice is considered "user-unfriendly", Bluetooth v4.2 introduced Cross-Transport Key Derivation (CTKD). CTKD allows two devices to pair once, either over BT or BLE, and generate both BT and BLE long term keys. Despite CTKD allowing ~~to cross~~ traversal of the security boundary between BT and BLE, the security implications of CTKD have not yet been investigated.

We present the first security analysis of CTKD and identify five cross-transport issues ~~for BT and BLE~~at the Bluetooth specification level. These issues enable, for the first time, exploitation of both BT and BLE by attacking either transport. Based on the identified issues, we demonstrate four novel cross-transport attacks resulting in device impersonation, traffic manipulation, and malicious session establishment. We refer to them 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~~details~~. We successfully demonstrate the BLUR attacks on 13 devices with 10 unique Bluetooth chips, and discuss effective countermeasures. We disclosed our findings and countermeasures to the Bluetooth SIG in May 2020.

1 Introduction

Bluetooth is a pervasive wireless technology used by billions of devices including mobile phones, laptops, headphones, cars, speakers, medical, and industrial appliances [13]. Bluetooth is specified in an open standard maintained by the Bluetooth special interest group (SIG)~~, and the~~ . The latest version of the standard is 5.2 [12]. The standard specifies two different, incompatible wireless transports, Bluetooth Classic (BT) and Bluetooth Low Energy (BLE). BT is best suited for high-throughput use cases, such as streaming audio and voice calls, while BLE is best suited for very low-power use cases such as localization and monitoring.

As BT and BLE were introduced at different ~~point~~ points in time to address different use cases, the standard maintains *separate security architectures and threat models* for BT [12, p. 947] and BLE [12, p. 1617]. While these security architectures address different threat models, they use similar security mechanisms, including pairing and secure session establishment. Pairing enables devices to establish a shared long term key, and secure session establishment enables paired devices to establish a secure communication channel by negotiating a session key that is derived from the pairing long term key.

Devices that support both BT and BLE have to pair twice to use both transports securely. ~~As pairing the same devices twice is considered "user-unfriendly",~~ 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 [12, p. 1401]. For example, two devices can pair over BT, generate the BT long term key, and then run CTKD to derive the BLE long term key (without having to pair over BLE). All major Bluetooth software stacks (Apple, Linux, Android, and Windows) and hardware providers (Cypress, Intel, Qualcomm, Broadcom, Apple, Sony, and Bose) implement CTKD. Apple presented CTKD as a core "always on" Bluetooth feature to improve usability [45].

~~CTKD is a promising attack target as it crosses the security boundary between BT and BLE (i.e., when using CTKD, pairing over one transport automatically provides security guarantees for both transports). Despite this fact, the security of CTKD remains unexplored. For example, the standard~~

~~does not include CTKD in the BT and BLE threat models and we are not aware of any security analyses of CTKD. So far, all existing attacks focused exclusively on either transport.~~

We present the first security analysis of CTKD, uncovering five standard-compliant security issues. ~~Our issues are novel because they are~~ Those issues are the first examples of cross-transport ~~issues~~ vulnerabilities for Bluetooth. Based on ~~those issues~~our findings, we demonstrate four cross-transport attacks, enabling ~~impersonation,~~ device impersonation, traffic interception, and ~~manipulation of traffic between victims~~traffic manipulation, as well as unintended device sessions. Our attacks ~~are standard-compliant and are thus effective against all devices that support CTKD. As we are the first to exploit CTKD and~~ enable BT and BLE cross-transport exploitation, ~~the attacks are orthogonal to other~~ are standard-compliant ~~attacks on BT and BLE [1, 3, 4, 10, 22, 23, 36, 39]~~and likely affect all devices supporting CTKD. We name our attacks ~~*BLUR*~~ BLUR attacks, as by exploiting CTKD they blur the security boundary ~~of~~ between BT and BLE.

In contrast to previously published attacks on BT and BLE [1, 3, 4, 10, 22, 23, 36, 39, 41, 44, 46], our attacks do not require the attacker to be present during pairing or secure session establishment. Therefore, our attacks have lower requirements for the attacker while still breaking BT and BLE security guarantees.

We implement the BLUR attacks using a widely available Bluetooth development board connected to a laptop running Linux and developing custom software based on open-source tools. This makes reproducing the BLUR attacks simple and affordable. Our evaluation demonstrates that all tested devices are vulnerable. We will release our tools to the public after the responsible disclosure process completes. We use our attack implementation to evaluate 13 devices, with 10 unique Bluetooth chips, from the major hardware and software vendors, e.g., Broadcom, Cambridge Silicon Radio (CSR), Cypress, Google, Intel, Linux, Qualcomm, and Windows and representing all Bluetooth versions that support CTKD (i.e., 4.2, 5.0, and 5.1) and even a device supporting Bluetooth version 4.1 to which CTKD has been backported.

We summarize our contributions as follows:

- We perform the first security analysis of CTKD (Section 3), and show that it enables ~~to cross~~ crossing the security boundary between BT and BLE. We identify five novel and very serious issues, ~~which enable the first~~ enabling cross-transport attacks between BT and BLE.

- We propose four attacks to exploit the issues in CTKD (Section 4). Our attacks allow impersonation, interception, traffic manipulation, and unintended sessions. ~~We~~ In Section 5, we present a low-cost implementation of the attacks based on a Linux laptop and a Bluetooth development board.

- We confirm that real-world BT and BLE devices are vulnerable to the BLUR attacks by evaluating our attacks on 13 unique devices (Section ~~??~~6). We provide ~~mitigation strategies to address the attacks directly in the Bluetooth standard. We have~~ concrete countermeasures to fix the presented issues.

We disclosed our findings and ~~mitigations~~ countermeasures to the Bluetooth SIG in May 2020. The Bluetooth SIG acknowledged our findings and assigned CVE-2020-15802 to the BLUR attacks. In September 2020, the Bluetooth SIG released a security note about our report at `https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/bluetooth-security/blurtooth/` (without contacting us).

2 Background

We now compare BT and BLE, and introduce CTKD.

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 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 devices, such as ~~laptop~~laptops, smartphones and tablets, provide 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 transports provide a pairing mechanism, named Secure Simple Pairing (SSP), to let two devices establish a long term key. During pairing, BLE allows negotiating the entropy of the long term key while BT does not. Both transports provide a secure session establishment mechanism to derive a session key from the long term key and protect the communication. During session establishment, BT allows negotiating the entropy of the session key while BLE inherits the entropy of the session key from the entropy of the long term key.

BT and BLE support a "Secure Connections" mode that uses FIPS compliant security primitives such as AES-CCM for authenticated encryption, ~~ECDH on~~ 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 ~~mechanisms~~ procedures that can be used to protect the pairing phase against man-in-the-middle attacks. Two examples of associations are "Just Works" that provides no pro-

tection and "Numeric Comparison" that provides protection against man-in-the-middle attacks by requiring user interaction (e.g., the user has to manually confirm that she sees the same numeric code on the pairing devices).

BT and BLE define master and slave roles in different ways. 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 after a radio link between the two is established. For BLE, master and slave roles are fixed and switching roles is not supported. The master acts as the connection initiator (BLE central) and the slave as the connection responder (BLE peripheral). High-end BLE devices, such as laptops and smartphones, implement both master and slave modes and are typically used as the master, while low-end devices, such as fitness trackers or smartwatches, typically implement only the slave mode.

2.2 Cross-Transport Key Derivation (CTKD)

Two devices that support BT and BLE have to pair over BT and over BLE to use both transports securely. Pairing the same two devices ~~two times is not user-friendly~~ twice is considered "user-unfriendly" and the Bluetooth standard ~~addressed this issue in Bluetooth~~ version 4.2 (released in 2014) ~~by introducing CTKD~~ introduces CTKD to address this issue. As shown in Figure 1, CTKD enables two devices to pair once, either over BT or BLE, and then securely use both [12, p. 280]. For example, a user can pair a headset and a laptop 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 CTKD to generate the BLE pairing key.

Before explaining ~~how does CTKDwork,~~ CTKD, it is important to review the differences between *pairable (bondable)* and *discoverable* states for BT and BLE. If a device is ~~pairable then is going to accept to pair with other devices, while~~ *pairable* then it will accept pairing requests from other devices. If if it is ~~discoverable is going to reveal his identity to other devices . It is widely believed that~~ *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 ~~to be able to pair, however only the pairable state is required~~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 ~~a pair of~~ headphones over BT, typically only the headphones are discoverable and pairable ~~and~~ while the laptop is only pairable. Hence, it is possible to pair with a device even if it is not discoverable [43].

The Bluetooth standard specifies the same CTKD function to derive BT and BLE long term keys. This function takes as inputs a 128-bit (16-byte) key and two 4-byte strings and derives a 128-bit (16-byte) key using AES-CMAC (see Section 5.3 for CTKD's internals). ~~What changes between BT and BLE are the inputs to the CTKD function.~~ CTKD for BT derives a BLE long term key (K_{BLE}) from a BT long term key (K_{BT}) and the strings "tmp2" and "brle". ~~While~~, while CTKD for BLE derives K_{BT} from K_{BLE} 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 key.

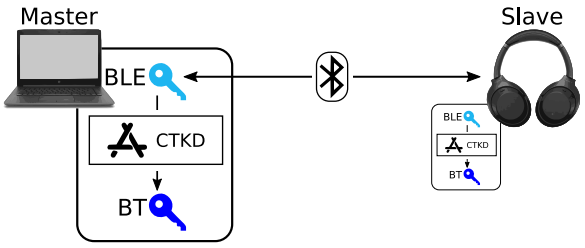

Figure 1: CTKD overview. CTKD is used by two devices who paired and share a long term key over BLE to derive a long term key for BT. CTKD can also be used to derive BLE pairing keys after two devices paired over BT.

CTKD is ~~widely supported by~~vendors such as broadly supported by, e.g., Apple [45], Google [5], Cypress [15], Linux [14], Qualcomm [34], and Intel [24]. CTKD is ~~used in combination~~ combined with "Secure Connections", ~~that is~~ a security mode that was introduced to enhance the security primitives of BT and BLE without affecting their security mechanisms. For example, "Secure Connections" ~~introduced~~ introduces AES-CCM authenticated-encryption for BT, and ECDH pairing for BLE.

3 Security Analysis of CTKD

~~BT and BLE are incompatible wireless technologies with different security architectures and threat models (see Section 2.1). CTKD, as shown in Figure 1, improves BT and BLE usability by allowing two devices to pair once, either over BT or BLE, and compute the keying material for both transports without requiring a second pairing (see Section 4).~~

~~CTKD is a critical attack target for several reasons. Firstly, CTKD crosses the security boundary between BT and BLE. Hence, if CTKD introduces a vulnerability on one transport, that vulnerability is exploitable for both BT and BLE. Secondly, CTKDis applicable to "Secure Connections", the most secure mode for BT and BLE. Hence, if CTKD is vulnerable then the attacker can break Bluetooth "Secure Connections".~~ 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.

~~Despite the potential security risks related to CTKD, the Bluetooth standard does not provide a security analysis of CTKD and does not include CTKD in the BT and BLE threat models [12, p. 1401]. We address this concern by performing the first security analysis of CTKD and uncovering cross-transport issues. A cross-transport issue enables an attacker to exploit BT from BLE or BLE from BT. This category of issues is novel in the context of Bluetooth. We now introduce our system and attacker models, and then we describe the identified cross-transport issues using a reference example.~~

3.1 System Model

Our system model considers two victims, Alice and Bob, who want to securely communicate over BT and BLE. Alice and Bob support CTKD and during pairing and session establishment ~~propose~~ select the strongest security mechanisms~~(e.g., SSP~~: Secure Simple Pairing (SSP), "Secure Connections", and "Numeric Comparison"~~). Such mechanisms~~. Those security procedures are expected to protect Alice and Bob against impersonation, eavesdropping, and man-in-the-middle attacks on BT and BLE~~.~~ [12, p. 269]. After completing pairing, Alice and Bob can run secure sessions ~~both~~ 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.

Regarding the notation, we indicate a BT ~~long term~~ pairing key with K_{BT}, a BT session key with SK_{BT}, a BLE ~~long term~~ pairing key with K_{BLE}, ~~and~~ a BLE session key with SK_{BLE}. ~~We~~ Moreover, we indicate a Bluetooth address with ADD~~. Furthermore, we indicate~~, 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.

3.2 Attacker Model and Goals

Our attacker model considers Charlie, a remote attacker in Bluetooth range with Alice and Bob. 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 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 ~~long term keys or session keys~~ any key shared between Alice and Bob and ~~does not observe Alice and Bob when~~ is not present while they pair or establish ~~a secure session. Regarding the attacker 's capabilities, the attacker~~ secure sessions. The attacker can scan and discover BT and BLE devices, jam the Bluetooth ~~spectrum, pair over~~

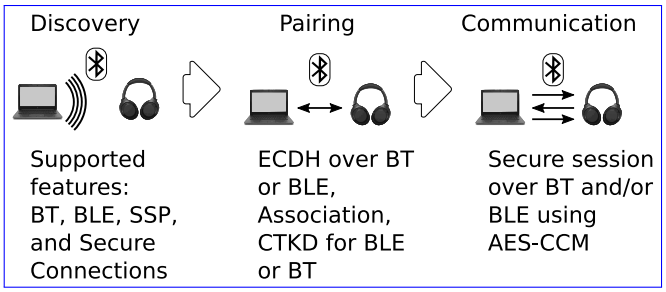

Figure 2: ~~Cross-transport issues~~ The three phases of the CTKD life cycle: Discovery (~~CTI~~to exchange features)~~with~~, Pairing (to agree on a pairing key and, through CTKD~~. We identify each issue with~~, create a ~~particular phase of~~ pairing key for the ~~Bluetooth secure connection life-cycle~~other transport), ~~but they are all directly~~ and Communication (to establish secure sessions on BT and/or ~~indirectly caused by the CTKD mechanism~~BLE).

~~BT and BLE using~~ channel, pair with Alice and Bob CTKD, propose weak association mechanisms (e.g., "Just Works"), and dissect and craft unencrypted Bluetooth packets.

The attacker has four goals. The first goal is to impersonate Alice (to Bob) and take over Alice's secure sessions. The second ~~goal~~ intent is to impersonate Bob (to Alice) and take over Bob's secure sessions. ~~Master and slave impersonations are two~~ Alice and Bob' impersonations are different goals as they require different ~~attacks~~attack techniques (i.e., Bluetooth master and slave impersonation attacks). The third ~~goal~~ objective is to establish a man-in-the-middle position in a secure session between Alice and Bob ~~. The third goal~~ and requires combining and synchronizing the impersonation attacks on Alice and Bob. The fourth goal is to pair and establish unintended and possibly stealthy sessions with Alice or Bob as an arbitrary device, without breaking ~~their secure session~~existing pairings and secure sessions between Alice and Bob.

3.3 CTKD Life Cycle

We first demonstrate the CTKD life cycle in a non-adversarial setting to later highlight the CTKD issues (Section 3.4) and attacks (Section 4). The first phase of the CTKD life cycle is *Discovery*, see Figure 2. During Discovery, Alice and Bob find 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., K_{BLE} or

K_{BT}) 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 Bob share a pairing key, then they complete a Bluetooth association phase. There are different association mechanism and in our threat model we assume that Alice and 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 K_{BT} from K_{BLE} or vice versa). The Bluetooth standard provides a CTKD function that is deterministic, as it uses a pairing key and constant strings (e.g., "brle" or "lebr") as inputs [12, p. 1401]. Moreover, the Bluetooth standard does not require to exchange any packet over the air to signal when CTKD is used and the outcome of its usage.

As soon as Alice and 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 pairing key and session nonces (e.g., SK_{BT} from K_{BT}, and SK_{BLE} from K_{BLE}), and uses the session key to encrypt and integrity-protect the link layer traffic with AES-CCM.

3.4 Cross-Transport Issues with CTKD

~~We now present~~

CTKD is 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 [12, 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 five ~~*cross-transport issues (CTI)* that we identified as a result of our security analysis of CTKD~~ cross-transport issues (summarized in Table 1). We now describe each issue in detail by using the CTKD life cycle phases presented in Section 3.3. ~~The order in which we present the issues follows the life-cycle of a Bluetooth connection, with discovery, pairing, and communication phases (see Figure ??). Section 4 then presents how to leverage the issues for attacks.~~

CTI	Name	Phase	Summary
1	Roles	Discovery	Role asymmetries
2	Sec. Conn.	Discovery	No Secure Connections
3	Association	Pairing	No uniform association
4	Key Overw.	Pairing	Overwrite pairing keys
5	States	Comm.	Pairable over BT and BLE

Table 1: 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 ~~the discovery phase~~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 Alice and 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*~~ a ***pairing request*** to Alice over BT even if Alice is expecting to receive only BT and BLE ~~*pairing responses*~~***pairing responses***.

CTI 2: Secure Connections (Discovery) ~~When~~ During Discovery, Alice and Bob ~~have discovered each other, they~~ exchange their capabilities before starting the pairing process. To use CTKD they declare "Secure Connections" support for the transport used for pairing~~(BT or BLE)~~. However, the specification does not specify if CTKD support requires "Secure Connections" support ~~*only for the pairing transport* or *for both transports*~~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~~ ***and*** BT "Secure Connections" but can also target devices that support BLE ***or*** BT "Secure Connections".

CTI 3: Association (Pairing) ~~After exchanging their capabilities~~During Pairing, Alice and Bob ~~perform the pairing process.~~ can pair either over BT or BLE. While BT and BLE pairings ~~are fundamentally different but they provide similar association mechanisms. However, the choice of the association mechanism is not enforced across BT and BLE~~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~~ re-pairing using "Just Works" pretending to be a ~~previously securely paired and~~ trusted device.

CTI 4: ~~States~~Key Overwrite (Pairing) ~~When~~ During Pairing, Alice and Bob ~~complete pairing they *remain pairable over BT and BLE* and Bob even~~ 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 ~~*remains discoverableover BLE*~~*CTKD will overwrite the existing pairing key*. This is an issue because an attacker who is impersonating either Alice or Bob can use CTKD to overwrite long term keys. For example, 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 Bob.

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 ~~paring~~pairing. This enables the attacker to establish unintended malicious sessions on both transports by pairing on the unused one and forcing CTKD.

~~**CTI 5: Key Overwrite** Once Alice and Bob are paired, they share a long term key, derive a second long term key via CTKD, and start a secure channel on BT and/or BLE. If Alice and Bob already shared a long term key for the transport used by CTKD then *CTKD will overwrite the existing key*. This is a serious issue because an attacker who is impersonating either Alice or Bob can use CTKD to overwrite long term keys. For example, 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 Bob.~~

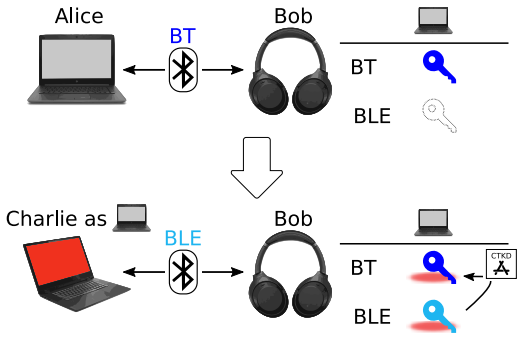

Figure 3: BLUR impersonation attack strategy. Charlie pairs with Bob over one transport (e.g., BLE) and (over)writes the pairing keys for both transports, ~~overwriting~~ including Alice's ~~key~~ BT pairing key.

4 BLUR Attacks on CTKD

We now design four novel CTKD cross-transport attacks based on the five cross-transport issues that we discuss in Section 3.4. ~~Our attacks are~~ We provide the first attacks that exploit CTKD~~by~~, blurring the security boundary between BT and BLE. ~~These~~ Our attacks are standard-compliant and enable impersonation, interception, and manipulation of traffic between victims, as well as unintended sessions with a victim device. We call our attacks *BLUR attacks*.

4.1 Master and Slave Impersonation

Figure 3 presents the BLUR impersonation attack strategy ~~using a slave impersonation attack as a reference example.~~ Before the attack takes place Alice and Bob (the victims) are running a secure BT session and they share a BT long term key (K_{BT}). As a side effect of CTKD, Alice and Bob are pairable on BLE. Charlie (the attacker), targets ~~the BLE transport~~ BLE (which is not used by the victims) and pairs with Bob over BLE ~~, pretending to be Alice~~ as Alice and triggers CTKD, while the real Alice is communicating with Bob over BT. Because of CTKD, ~~Bob overwrites the BT long term~~ Charlie forces Bob to overwrite the BT pairing key that he established with Alice with ~~the one derived when pairing with Charlie~~his own. As a result, Charlie takes over Alice's BT session ~~, and~~ from BLE. The real Alice can no longer connect to Bob as she does not possess the correct K_{BT}.~~~~

~~In the following two paragraphs we describe the technical details of the BLUR~~ 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 attacks ~~.~~as follows:

Master impersonation Charlie impersonates Alice (master) and takes over her BT secure session with Bob as in Figure 4. Charlie discovers Bob as he is pairable over BLE

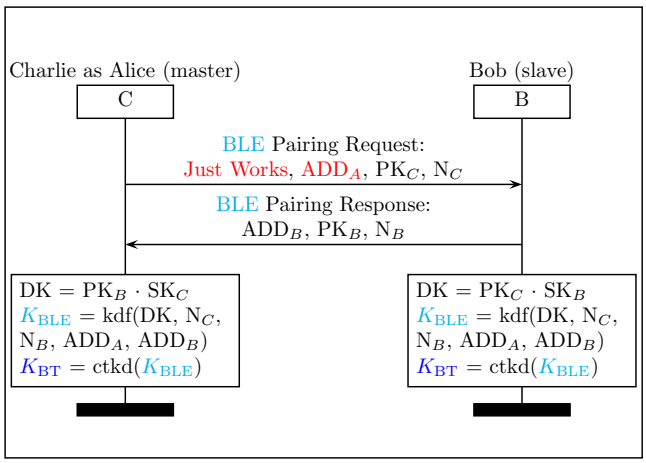

Figure 4: Master impersonation attack and takeover. Charlie (acting as master) pairs with Bob over BLE, overwriting Alice's key.

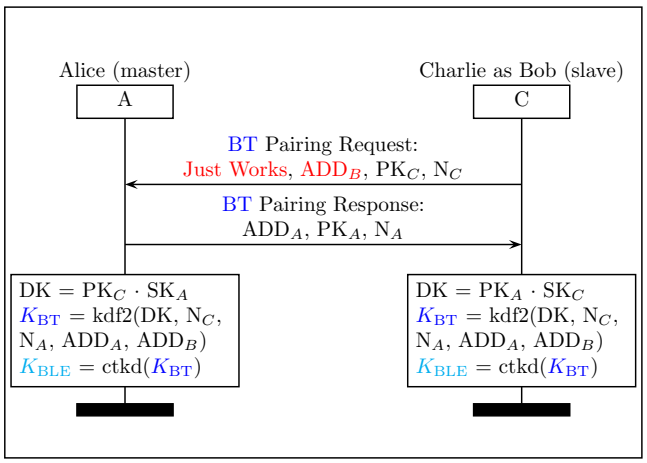

Figure 5: Slave impersonation attack and takeover. Charlie (acting as slave) sends a BT pairing request to Alice (master) as Bob, overwriting Bob's key.

and sends a BLE pairing request using Alice's Bluetooth address (ADD$_A$)~~and requests to use~~ , Secure Connections support (to trigger CTKD), and "Just Works" association to avoid user interaction. ~~The~~ Charlie's BLE pairing request ~~is standard-compliant because Charlie impersonates a BLE master and is going to be accepted by Bob who is pairable over BLE . Bob sends~~ 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 a BLE pairing response believing that Alice wants to pair (or ~~repair~~re-pair) over BLE using CTKD. Charlie and Bob use the exchanged ~~nonces and~~ public keys to compute DK. Then they use DK and the exchanged nonces (NC, N*B*) to compute K$_{BLE}$ ~~(kdf) and derive~~ . Then, they locally compute K$_{BT}$ from K$_{BLE}$ using the CTKD's key derivation ~~functions~~ function (ctkd). As a result of the master impersonation attack, Charlie forces Bob to overwrite the BT pairing key that he established with Alice with his BT pairing key, establishes a BLE pairing key ~~, shares a BLE key~~ with Bob, and takes over Alice's BT session. ~~Alice can no longer establish secure sessions with Bob as, during pairing with Charlie, Bob overwrote her shared key.~~

Slave impersonation Charlie impersonates Bob (slave) and takes over his BT secure session with Alice as in Figure 5. ~~Charlie sends a BT pairing request using~~ 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 BT and BLE 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 address (ADD$_B$) and ~~requests to use~~ "Just Works" association to avoid user interaction. ~~The BT pairing request is standard-compliant because BT allows a slave to send a BT pairing request by switching role before pairing is started.~~

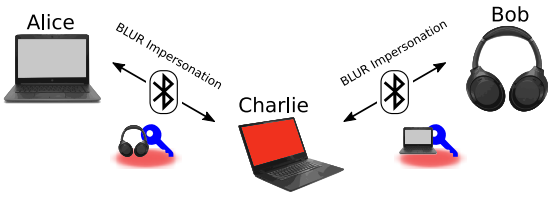

Figure 6: BLUR man-in-the-middle attack. The attacker uses the BLUR Impersonation attack against two devices that were previously paired. The two devices do not detect a change but Charlie now has access to all traffic.

Alice, who is pairable over BT, sends a BT pairing response believing that ~~she is talking to Bob (a trusted device)~~Bob wants to re-pair over BT using CTKD. Charlie and Alice use the exchanged ~~nonces and~~ public keys to compute DK. Then they use DK and the exchanged nonces to derive K$_{BT}$ (kdf2)~~, and~~ . Then they locally derive K$_{BLE}$ from K$_{BT}$ using CTKD's key derivation functions (ctkd). 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 ~~possess the correct paring~~ possesses the correct pairing keys.

As summarized in Table 2, the master impersonation attack takes advantage of all the cross-transport issues that we present in Section 3.4 except CTI 1. In particular, the attacker takes advantage of non-consistent "Secure Connections" support (CTI 2), lack of consistency between BT and BLE asso-

	CTI 1 Roles	CTI 2 ~~SC~~ Sec. Conn.	CTI 3 ~~Association~~ Assoc.	CTI 4 ~~States~~ Key ~~(KO~~
Master Imp.	x	✓	✓	✓
Slave Imp.	✓	✓	✓	~~x~~ ✓
MitM	✓	✓	✓	✓
Unin. Sess.	x	✓	x	x

Table 2: ~~The mapping between~~ Mapping the ~~requirements~~ of our four BLUR attacks to the discovered cross-transport issues (CTI)~~identified in Section 3.4 and the four BLUR attacks discussed in Section 4.We abbreviate "Secure Connections" with SC, and Key Overwrite with KO.~~

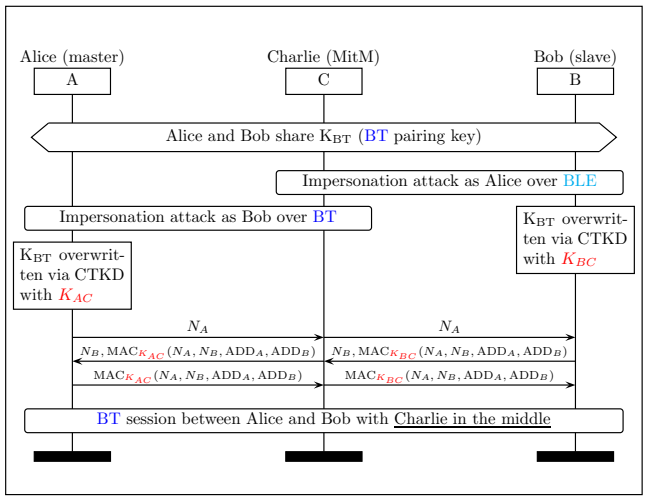

Figure 7: MitM attack and takeover. Charlie impersonates Alice as in Figure 4, impersonates Bob as in Figure 5, let the victims mutually authenticate and then gets access to their traffic.

ciation methods (CTI 3), more opportunities to pair (CTI ~~4~~5), and key overwriting (CTI ~~5~~4). The slave impersonation attack takes advantage of all CTIs except CTI ~~4~~5, including the role asymmetries between BT and BLE (CTI 1).

4.2 Man-in-the-Middle

Figure 6 presents the high-level description of our BLUR man-in-the-middle attack. As in the previous section, Alice and Bob are paired over BT and they run a secure session over BT. During this attack, Charlie sequentially performs the master and slave impersonation attacks described in Section 4.1. As a result, the attacker overwrites Alice and Bob's BT pairing keys with known keys, establishes BLE long term keys with Alice and Bob, and positions himself in the middle to access all traffic between the victims and to inject valid traffic both on BT and BLE.

Figure 7 shows the details of the MitM attack. Firstly, Charlie impersonates Alice to Bob over BLE (as in Figure 4), overwrites Bob's BT key with his key (K_{BC}). Secondly, Charlie impersonates Bob to Alice over BT as in Figure 5 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, Alice and Bob establish 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 2, 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 3.4.

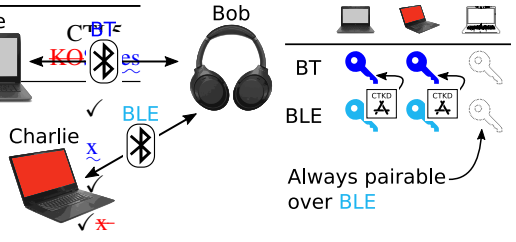

Figure 8: BLUR unintended sessions attack. Charlie sends a BLE pairing request to Bob (who remains pairable over BLE due to CTKD) as an unknown device with arbitrary capabilities. After CTKD completes, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking Bob's existing pairings and sessions.

4.3 Unintended Sessions

Figure 8 presents a BLUR unintended session attack targeting Bob. In this scenario, Alice and Bob are running a secure session over BT but they are still pairable over BLE in order to accept pairing requests with other devices and run CTKD. Charlie targets Bob (slave) by sending him a ~~paring~~ pairing request over BLE as an unknown device. Charlie can pretend to be any device having arbitrary capabilities, e.g., Bluetooth address, Bluetooth name, device class, "Secure Connections" support, and weak association. Bob, accepts to pair with Charlie while continuing his session with Alice. Then, Charlie and Bob negotiate K_{BLE}, and derive K_{BT} using CTKD. Now, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking his existing pairings or sessions with other devices (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 [12, p. 1401]. Then, Alice and Charlie compute K_{BLE},

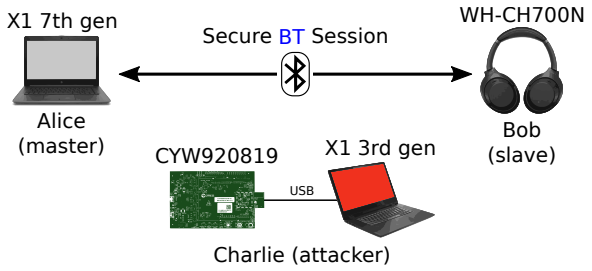

Figure 9: Example BLUR Attack Scenario. Alice (master) is a ThinkPad X1 7th gen, Bob (slave) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in absence of Charlie, and are running a secure BT session.

and derive K_{BT} using CTKD. Now, 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 Alice and Bob in many ways. For example, he can use the session to drop known exploits such as BlueBorne [6], BLEEDINGBIT [7], or SweynTooth [20], new exploits, and to enumerate and tamper with BT and BLE services and characteristics (including the protected ones).

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 2 the unintended session BLUR attack takes advantage of CTI 2 and CTI 4.

5 Implementation

In this section we describe our attack scenario, our ~~attack device, an implementation of~~ implementation of a custom attack device to perform the BLUR attacks ~~(proposed in Section 4) using open-source software and off-the-shelf hardware, an implementation of the CTKD mechanism used to validate our attacks, and an evaluation of our attacks on unique devices from different hardware and software vendors~~and our re-implementation of CTKD's key derivation function. The tools that we developed will be open-sourced~~after responsible disclosure with the Bluetooth SIG.~~

5.1 Attack Scenario

Our attack scenario ~~is presented~~ 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 ~~(Alice, master) and~~ and Bob by a pair of Sony WH-CH700N headphones~~(Bob, master)~~. The attacker (Charlie) uses a CYW920819 development board [16] and a 3rd generation ThinkPad X1 laptop ~~. As in our attack descriptions in Section 4, the victims have securely paired in absence of Charlie, and are running a secure BT session. The evaluation results~~ as an attack device. The implementation of the attack device is presented in Section ~~6.2 are obtained by using~~ 5.2. In our evaluation, we use the same attack scenario ~~targeting~~ with different victim devices.

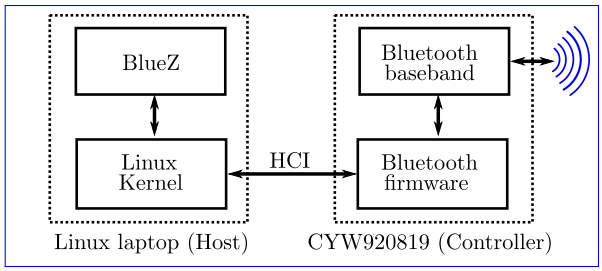

Figure 10: Attack Device Block Diagram. The attack device is composed of Linux laptop (Host) and a CYW920819 (Controller) connected via USB and communicating using the Host Controller Interface (HCI) protocol.

~~Table 3 presents the relevant Bluetooth features supported by Alice and Bob~~To understand the capabilities of the victims and the attacker we summarize their most important Bluetooth features in Table 3. We note that Bob is 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~~in Section 3.4.~~. Furthermore to conduct the attacks we had to develop an attack device that enabled us to change all the features in Table 3~~also shows the featuressupported by Charlie, and indicates with an asterisk (*) the features that we can modify with our implementation of the BLUR attacks. For example, our implementation enables to send pairing requests over BT and BLE with with arbitrary Bluetooth addresses, names, associations, "Secure Connections" (SC) support, and authentication requirements (AuthReq)~~. 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.

5.2 Custom Attack Device

~~Our attack device~~

To implement the BLUR attack we had to develop a custom attack device. As we can see from its block diagram in Figure 10, the attack device consists of a Linux laptop ~~(Bluetooth host) connected~~ implementing the Bluetooth host component using BlueZ (i.e., user-space) and the Linux

	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*~~ 8716
Version	5.1	4.1	5.0 ~~*~~
Name	x7	WH-CH700N	~~x1*~~ x1
ADD	Redacted	Redacted	~~Redacted*~~ Redacted
Class	0x1c010c	0x0	~~0x0*~~ 0x0
BT SC	True	Only Controller	~~True*~~ True
BT AuthReq	0x03	0x02	~~0x03*~~ 0x03
BLE SC	True	True	~~True*~~ True
BLE AuthReq	0x2d	0x09	~~0x2d*~~0x2d
CTKD	True	True	~~True*~~ True
h7	True	False	~~True*~~ True
Role	Master	Slave	~~Master*~~ Master
IO	Display	No IO	~~Display*~~ Display
Association	"Numeric C."	"Just Works"	"Numeric C." ~~*~~
Pairable	True	True	~~True*~~ True

Table 3: 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.~~We append an asterisk (*) to the attacker's features that we can modify with our implementation.~~

kernel. The laptop is connected via USB to a CYW920819 development board~~(Bluetooth controller). We implement our attack device by developing custom code and tools both for Linux and the board . Regarding~~ . 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.

For the host, we ~~modify and recompile the Linux kernel and BlueZ according to our needs. For example, by changing the kernel we enable parsing of diagnostic messages from the controller, and by changing BlueZ we can develop custom user-space management commands for~~ used standard Linux tools to configure an 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. Furthermore, we wanted to access the traffic exchanged over the air by our attack device. This is not available on a standard Bluetooth device. To achieve this goal we sent a proprietary HCI command from the host to enable diagnostic mode on 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 host without requiring over-the-air BT and BLE sniffers.

~~Regarding the controller ,~~ we use the Modifying the controller required us to interact directly with the development board's ~~proprietary patching mechanism to modify the Bluetooth firmware according to our needs. For example, by writing the firmware 's RAM we can change the attack device's features, including the features containing an asterisk (*)in Table 3. This process required significant engineering effort as we had to dump the Bluetooth firmware from the board, reverse-engineer the relevant~~ Bluetooth firmware. To extract the firmware we used a proprietary HCI command from Cypress to read and save a 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 from the board's SDK to extract the various segments from the snapshot including the ROM, the RAM, and the scratchpad segments. 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 our first reverse-engineering pass we isolated the libc functions (e.g., `malloc` and `calloc`) by looking at the signatures and the code patterns of the functions that are called the most. Then, we found the firmware debugging symbols in the board's SDK and loaded them into Ghidra. Using the debugging symbols we isolated functions and data structures ~~, and write and test our ARM assembly patches .~~

~~Tool Usage `ghidra` RE the devboard firmware [40] `internalblue` Patch devboard firmware [31]`wireshark` Monitor HCI, LMP, and SMP `hciconfig` Configure HCI interfaces `hcitool` Scan, connect and enumerate BLE devices `bleah` Scan, connect and enumerate BLEdevices `scapy` Craft and decode packets [11] `pybt` Custom BLEpairing [37] `linux414` Modify BLE pairing capabilities `bluez` Modify Linux userspace configuration `pybluez` Test BT~~ relevant for the BLUR attacks. Then, we wrote assembly patches to change their behaviors and we apply those patches at runtime using internalblue [31] . 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 3).

5.3 Re-Implementing CTKD

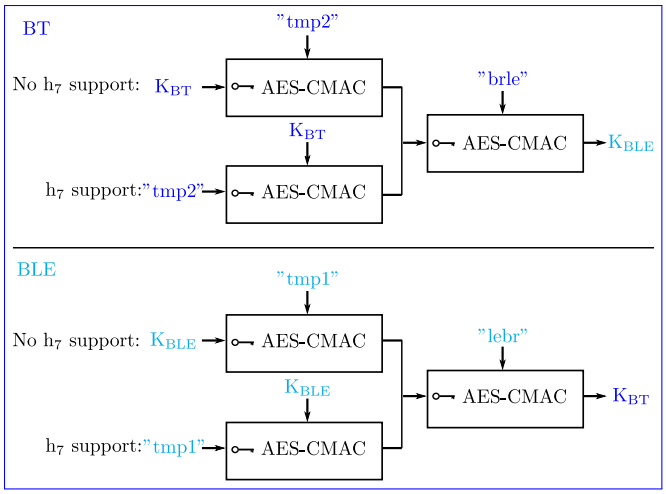

Figure 11: CTKD function for BT (top) and BLE (bottom). The functions are the same but use a sequence of two AES-CMAC with different input quantities. In the first AES-CMAC, the devices use a constant string as key and the pairing key as input if they support the h7 conversion function, otherwise, they 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.

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~~using the BlueZ API `scapy` Configure HCI, manage BT and BLE sockets `bluetoothctl` Manage, pair and connect devices~~. 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 [12, p. 1401]. Our implementation uses the PyCA cryptographic module [8], 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 now describe the CTKD key derivation function implementation details. The Bluetooth standard specifies a single CTKD function (see Section 2.2) that is used with different parameters for BT and BLE. Figure 11 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 [18]. In particular, BT uses K_{BT}, `"tmp2"` and `"brle"` and derives K_{BLE}, while BLE uses K_{BLE}, `"tmp1"` and `"btmgmt lebr~~Manage, pair and connect devices Open-source tools used to implement the BLUR attacks.~~"` and derives K_{BT}.

~~Our attack device makes use of several free and open-source tools to automate the configuration and management of BT, BLE, and~~ In the first AES-CMAC, if both devices support the h7 conversion function in the Bluetooth standard [12, 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. In the second AES-CMAC, the ~~BLUR attacks. Table ?? presents the list of such tools with a brief description of their usage. Overall, our usage of low-cost hardware and open-source software will enable other researchers to easily reproduce the BLUR attacks.~~ 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.

6 Evaluation

In this section we present how we conducted the BLUR attacks and our evaluation results on 13 unique devices (see Table 4). The tested devices represent popular laptops, phones, headphones, and an embededd platform. The devices are from a broad set of device producers (Samsung, Dell, Google, Lenovo, and Sony), run different operating systems (Android, Windows, Linux, and proprietary OSes), use different Bluetooth chipsets (from Broadcom, CSR, Cypress, Intel, Qualcomm, and Samsung).

6.1 Realizing the Attacks

The BLUR attacks, presented in Section 4, include master impersonation, slave impersonation, man-in-the-middle, and unintentional session attacks. In the next paragraphs, we describe how we ~~implemented them based on our attack device in the attack scenario presented in Section 5.1~~conducted them using our custom attack device described in Section 5.2.

Laptop (master) impersonation attack To impersonate the laptop, we configure our attack device to clone the laptop Bluetooth features, including Bluetooth address, Bluetooth name, device class, BT and BLE "Secure Connections" support, and advertised services. We accomplish this task by patching 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.

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, 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.

Headphones (slave) impersonation attack To impersonate the headphones, we configure our attack device to clone the headphones Bluetooth features using the same technique adopted for the laptop impersonation. Once 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 using `btmgmt`'s TUI. The laptop accepts to pair over BT, updates the BT long term key, and runs CTKD for BLE. Then, we establish a secure BT session with the headphones.

To evaluate master and slave impersonation attack experimentally, we used the attack device both as the attacker and as one of the victims. 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 master victims.

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 attack device establishes new BT and BLE keys both with the headphones and the laptop and starts unintended sessions with both over BT and BLE.

~~The Bluetooth standard does not provide a reference implementation for the key derivation function used by CTKD, and provides limited documentation about its design [12, p. 1401]. We decided to implement it in Python 3 using the PyCA cryptographic module [8] and we successfully tested our implementation against the test vectors in the standard. We used our implementation to validate the BT and BLE keys derived using CTKD while performing our attacks and the code will be open-sourced. We now describe the CTKD key derivation function implementation details.~~

~~CTKD key derivation function for BT (top) and BLE (bottom).~~

~~As explained in Section 2.2, the Bluetooth standard specifies a single CTKD function that is used with different parameters for BT and BLE. Figure 11 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 [18]. In particular, BT uses , "`tmp2`" and "`brle`" and derives , while BLE uses , "`tmp1`" and "`lebr`" and derives .~~

~~In the first AES-CMAC, if both devices support the h7 algorithm, 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. In the second AES-CMAC, the 16-byte output of the first AES-CMAC is used as key and the string as input. The 16-byte output of the second AES-CMAC is the derived long term key.~~

~~With our attack implementation (Section 6.1), we are capable of conducting all four BLUR attacks. We used the attack device both as the attacker and as one of the victims. 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 practical because it allows us to quickly test many slave victims. For the slave impersonation, we use the same procedure and quickly test many master 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. Regarding the unintended session attack, we test~~ 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

We evaluated the BLUR attacks on 13 devices, ~~and~~ the results are summarized in Table 4~~shows our evaluation results~~. The first six columns indicate the device producer, device model, OS, chip manufacturer, chip model, and supported Bluetooth

Device			Chip		Bluetooth	BLUR Attack		
Producer	Model	OS	Producer	Model	Version	Role	MI/SI	MitM
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 4: BLUR attacks evaluation results. The last three columns contain a checkmark (✓) if a device is vulnerable to the master ~~Impersonation~~ 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 ~~it~~ 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 13 devices (10 unique Bluetooth chips) are vulnerable to ~~the~~ all relevant BLUR attacks.

version. The seventh column indicates the attacker role. The last three columns contain a checkmark (✓) if a device is vulnerable to the master ~~Impersonation~~ impersonation attack (MI), slave impersonation attack (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 13 devices (10 unique Bluetooth chips) that we tested are vulnerable to ~~the~~ all relevant BLUR attacks.

~~Our list of vulnerable devices is from a broad set of device producers (Samsung, Dell, Google, Lenovo, and Sony), operating system producers (Android, Windows, Linux, and proprietary OSes), and Bluetooth chip producers (Broadcom, CSR, Cypress, Intel, Qualcomm, and Samsung). Our~~ As we tested a wide range of devices that were all vulnerable, our evaluation demonstrates that the BLUR attacks are practical, standard-compliant, and ~~affects~~ affect all the Bluetooth versions that support CTKD~~(i.e., Bluetooth versions $\geq$ 4.2)~~. 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 ~~to~~ fix the issues that we uncover in CTKD and we provide our set of countermeasures for the Bluetooth standard in Section 7.2.

7 Discussion

We now discuss the lessons learned ~~,~~ and our set of countermeasures to mitigate the BLUR attacks.

7.1 Lessons Learned

~~One key lesson~~ There are several lessons that we learned while analyzing CTKD and ~~designing, implementing and~~ developing the BLUR attacks. In this section we report those lessons evaluating the BLUR attacks~~is that designers of security mechanisms must be careful when bridging~~. 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.

Cross-transport mechanisms need a cross-transport threat model Security mechanisms, such as CTKD, that cross the security boundary between two technologies with different ~~security architectures and threat models such as BT and BLE. As demonstrated in this work, such a combination can introduce~~ threat models should be designed using a cross-transport ~~issues that can be exploited on a large scale.~~

~~Another key lesson that we learned is that isolating cross-transport issues is challenging, as such issues manifest at the security boundary between the affected transports and usually they are not part of~~ threat model. For example, the Bluetooth standard should consider that an attacker might try to exploit BT from BLE via CTKD and vice versa. Unfortunately, at the time of writing, the Bluetooth standard lacks a cross-transport threat model. ~~Separate security analyses of the affected transport are insufficient to discover cross-transport issues . Such issues require a security analysis that considers both transports and related threat models at the same time. In particular, the analysis must take into account an attacker with cross-transport capabilities.~~ 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.

Similar security mechanisms with different threat models do not provide the same security guarantees 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 mechanisms are 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., ~~an attacker who can take advantage of weak mechanisms on one transport to exploit both~~the BLUR attacks).

Properly weighting usability against security benefits is key CTKD was introduced to improve 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 mechanism.

7.2 Countermeasures

We now present ~~our~~ a set of countermeasures ~~for the BLUR attacks. We recommend addressing the BLUR attack at the Bluetooth standard level, as the BLUR attacks are standard-compliant.~~ to address all the five cross-transport issues (CTI) that we present in Section 3.4. Our countermeasures can be implemented in the device's Bluetooth Host (~~implemented in the device~~OS) ~~by storing~~ i.e., device's OS), by storing and checking extra metadata about ~~a trusted Bluetooth device and by using available HCI commands and events~~its state and trusted remote devices.

~~**Disable CTKD key overwrites**~~**Align BT and BLE roles (CTI 1)** ~~CTKD allows to write and overwrite BT long term keys from BLE and BLE long term keys from BT. This enables an attacker to impersonate a device and take over her existing session on one transport by attacking the other~~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, a device should ~~disallow key overwrites with CTKD when a paired device wants to re-pair. For example, re-pairing over BT should not overwrite a BLE long term 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~~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) 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 BT and BLE and 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". ~~To fix this issue, a~~ A device should keep track of ~~which BT and BLE keys are established using CTKD, record the~~ the remotes' strongest association mechanism used while pairing and enforce it for subsequent (re-)pairings.

~~**Enforce Secure ConnectionsIn our experiments, we can use CTKD with the WH-CH700N headphones even if they only support "Secure Connections" for BLE**~~ **Disable CTKD key overwrites (CTI 4)** CTKD allows (over)writing BT long term keys from BLE and vice versa. This enables an attacker to impersonate a device and take over her existing session on one transport by attacking the other. ~~This should not happen as CTKD should be used only when "Secure Connections" is supported on both BT and BLE.~~ To fix this issue, a device should ~~enforce that "Secure Connections" is supported on~~ disallow key overwrites with CTKD when a paired device wants to re-pair. For example, re-pairing over BT should not overwrite a BLE long term 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 when not needed (CTI 5) In our experiment we confirmed that a device might remain pairable over BT and BLE ~~before running CTKD and raise an error if this is not the case.~~

~~**CTKD Notifications** CTKD is transparent to end-users and is specified in the standard as an optional feature. We exploit those facts to improve the stealthiness of our attacks. Given that CTKD is~~ 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 issue, a ~~security-critical feature we believe that it should not be considered optional, and a~~ device should ~~notify the user every time the feature is used~~automatically stop being pairable on a transport that is not currently in use. For example, ~~the device should notify the user when she is re-pairing with a trusted device and is using CTKD to overwrite a long term key~~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 pairing mode.

8 Related Work

~~Bluetooth~~ The Bluetooth provides a royalty-free and widely-available cable replacement technology [21] . 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 [25, 30]. Standard-compliant attacks on BT include attacks on legacy pairing [38], secure simple pairing (SSP) [10, 22, 39], Bluetooth association ~~[23]~~ [23, 41], key negotiation [1], and authentication procedures [3, 29, 42]. Standard-compliant attacks on BLE include attacks on legacy pairing [36], key negotiation [4], SSP ~~[10],~~ [10, 46] , reconnections [44], and GATT [26]. 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 [6] and BLE [7, 20]. 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 [17, 32, 33] but ~~none~~ neither of those surveys ~~(and~~ nor the Bluetooth standard ~~) is considering~~ 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 [9] and Google Nearby Connections [2]. 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 [19] and it is considered relatively weak [33]. SAFER+, used for authentication, was analyzed as well [28]. BT and BLE "Secure Connections" use the AES-CCM authenticated-encryption cipher. AES-CCM was extensively analyzed [27, 35] and it is FIPS compliant. Our BLUR attacks target key negotiation and not cryptographic primitives, and are effective even with perfectly secure cryptographic primitives.

9 Conclusion

~~In this work, we~~ We present the first security ~~analyses~~ analysis of CTKD and identify novel standard-compliant and cross-transport issues and attacks against BT and BLE. Our attacks show that CTKD enables an attacker to cross the security boundary between BT and BLE. ~~These wireless transports have different security architectures and threat models. Despite this fact, the Bluetooth standard does not include CTKD in the~~ 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 ~~threat models and the security implications of CTKD are not well understood~~security guarantees.

We identify five cross-transport issues related to roles (CTI 1), "Secure Connections" ~~, association, device states, and key overwrite . Using the~~ (CTI 2), association (CTI 3), key overwrite (CTI 4), and pairing states (CTI 5). Based on those issues, we ~~design and implement novel cross-transport~~ develop attacks against BT and BLE enabling ~~impersonation~~impersonations, traffic manipulation, and malicious session establishment. ~~Our standard-compliant attacks exploit BT and BLE just by targeting one of the two.~~ We name our attacks BLUR attacks as they blur the security boundary between BT and BLE.

We provide and discuss a low-cost implementation of the BLUR attacks using off-the-shelf hardware and open-source software. To demonstrate that our attacks are practical, we ~~use our implementation to successfully attack~~ successfully exploit 13 devices from different hardware and software manufacturers. Our devices range across all the Bluetooth versions supporting CTKD (~~version~~ e.g., versions greater or equal to 4.2) and also a version of Bluetooth 4.1 ~~. As the BLUR attacks are standard-compliant, all devices supporting CTKD are potentially vulnerable~~with backported CTKD features.

~~We sketch a set of countermeasures to address the BLUR attack directly in~~

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., 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 ~~. The countermeasures require to keep additional state about paired devices. We have disclosed our findings and our~~ level. We responsibly disclosed our vulnerabilities, attacks, and countermeasures to the Bluetooth SIG~~in May 2020.~~ .
=10000

References

[1] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. The KNOB is broken: Exploiting low entropy in the encryption key negotiation of Bluetooth BR/EDR. In *Proceedings of the USENIX Security Symposium*. USENIX, August 2019.

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

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

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

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

[6] Armis Inc. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/, Accessed: 2018-01-26, 2017.

[7] Armis Inc. BLEEDINGBIT: The hidden attack surface within BLE chips. https://armis.com/bleedingbit/, Accessed: 2019-07-24, 2019.

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

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

[10] Eli Biham and Lior Neumann. Breaking the bluetooth pairing–fixed coordinate invalid curve attack. http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf, Accessed: 2018-10-30, 2018.

[11] Philippe Biondi. Scapy: Packet crafting for python2 and python3. https://scapy.net/, Accessed: 2018-01-26, 2018.

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

[13] Bluetooth SIG. Bluetooth Markets. https://www.bluetooth.com/markets/, Accessed: 2019-10-23, 2019.

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

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

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

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

[18] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf, 2018. Recommendations of the NIST.

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

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

[21] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf J Joeressen, and Warren Allen. Bluetooth: Vision, goals, and architecture. *ACM SIGMOBILE Mobile*

Computing and Communications Review, 2(4):38–45, 1998.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[37] Mike Ryan. Pybt: Hackable bluetooth stack in python. https://github.com/mikeryan/PyBT, Accessed: 2019-06-19, 2015.

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

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

[40] National Security Agency USA. Ghidra: A software reverse engineering (SRE) suite of tools developed by NSA's research directorate in support of the cybersecurity mission. https://ghidra-sre.org/, Accessed: 2019-02-04, 2019.

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

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

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

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

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

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