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

Daniele Antonioli[*]
EURECOM
Biot, France
daniele.antonioli@eurecom.fr

Nils Ole Tippenhauer
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
tippenhauer@cispa.de

Kasper Rasmussen
University of Oxford
Oxford, UK
kasper.rasmussen@cs.ox.ac.uk

Mathias Payer
EPFL
Lausanne, Switzerland
mathias.payer@nebelwelt.net

ABSTRACT

Bluetooth is a pervasive wireless technology specified in an open standard. The standard defines Bluetooth Classic (BT) for high-throughput wireless services and Bluetooth Low Energy (BLE) very low-power ones. The standard also specifies security mechanisms, such as pairing, session establishment, and cross-transport key derivation (CTKD). CTKD enables devices to establish BT and BLE security keys by pairing just once. CTKD was introduced in 2014 with Bluetooth 4.2 to improve usability. However, the security implications of CTKD were not studied carefully.

This work demonstrates that CTKD is a valuable and novel Bluetooth attack surface. It enables, among others, to exploit BT and BLE just by targeting one of the two (i.e., Bluetooth cross-transport exploitation). We present the design of the first cross-transport attacks on Bluetooth. Our attacks exploit issues that we identified in the specification of CTKD. For example, we find that CTKD enables an adversary to overwrite pairing keys across transports. We leverage these vulnerabilities to impersonate, machine-in-the-middle, and establish unintended sessions with any Bluetooth device supporting CTKD. Since the presented attacks blur the security boundary between BT and BLE, we name them *BLUR attacks*. We provide a low-cost implementation of the attacks and test it on a broad set of devices. In particular, we successfully attack 16 devices with 14 unique Bluetooth chips from popular vendors (e.g., Cypress, Intel, Qualcomm, CSR, Google, and Samsung), with Bluetooth standard versions of up to 5.2. We discuss why the countermeasures in the Bluetooth are not effective against our attacks, and we develop and evaluate practical and effective alternatives.

[*]This work started while Daniele was at EPFL.

CCS CONCEPTS

• **Security and privacy** → **Systems security**; **Network security**; **Mobile and wireless security**; **Security protocols**;

KEYWORDS

CTKD, Bluetooth, Bluetooth Classic, Bluetooth Low Energy

ACM Reference Format:
Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. 2022. BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30-June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3488932.3523258

1 INTRODUCTION

Bluetooth is a pervasive wireless technology used by billions of devices, including mobile phones, laptops, headphones, cars, speakers, medical and industrial appliances [11]. It is specified in an open standard maintained by the Bluetooth special interest group (SIG), and its latest version is 5.3 [14]. The standard specifies two transports: *Bluetooth Classic (BT)* and *Bluetooth Low Energy (BLE)*. BT is best suited for connection-oriented and high-throughput use cases, such as streaming audio. BLE is optimized for connection-less and very-low-power use cases such as fitness tracking.

The Bluetooth standard defines dedicated security architectures and threat models for BT [14, p. 947] and BLE [14, p. 1549]. Each transport provides a *pairing* and a *session establishment* protocol. Pairing lets two devices agree upon a long-term pairing key acting as a root of trust. Session establishment allows paired devices to establish a secure channel through a fresh session key derived from their pairing key.

Traditionally, two devices supporting BT and BLE would have to pair separately to use BT and BLE securely. In 2014, the Bluetooth standard (v4.2) introduced *Cross-Transport Key Derivation (CTKD)* to address this usability issue. CTKD enables pairing over BT or BLE, generating a pairing key for that transport, and deriving the pairing key for the other transport without pairing a second time [14, p. 1366]. As a result, CTKD tries to unify BT and BLE pairing, but its security implications are still unclear.

We uncover that CTKD provides a valuable and novel Bluetooth *attack surface*. In particular, CTKD allows exploiting BT from BLE and vice versa, which is an unprecedented threat for Bluetooth.

Moreover, attacks on CTKD are portable across targets as they exploit a security protocol in the Bluetooth standard. Additionally, since CTKD is used in conjunction with Bluetooth's most secure modes (e.g., Secure Simple Pairing and Secure Connections), exploiting it would defeat even the strongest Bluetooth setups. Finally, stopping and detecting an attack on CTKD is not trivial as it is transparent to end-users.

Despite CTKD being a critical attack surface, it has not received any attention from the research community and only limited care from the Bluetooth standard. For versions 4.1, 4.2, and 5.0, the Bluetooth standard does not provide any security argument about CTKD. In version 5.1, the standard introduced some recommendations about CTKD key overwrite attempts. In particular, a device should not overwrite a key if the overwritten key has higher entropy or machine-in-the-middle (MitM) protection. Other attacks on CTKD are not discussed (e.g., attacks not requiring to overwrite keys or to overwrite keys but without downgrading their strength or MitM protection).

We present a novel family of *cross-transport* attacks for Bluetooth. Unlike prior work [2, 4, 5, 9, 24, 25, 39, 43, 44, 47, 49] our attacks are the first attacks targeting CTKD and the first sample of cross-transport attacks for Bluetooth (i.e., attacks capable of exploiting BT and BLE just by targeting one of the two). Prior attacks focused either on BT or BLE. For a detailed comparison, see Section 9.

Our cross-transport attacks achieve *impactful* goals. In particular, they enable impersonation and MitM attacks on any BT and BLE device. Furthermore, they allow an attacker to establish unintended and anonymous BT and BLE sessions with a victim. As a result the adversary, among others, can access sensitive data and inject arbitrary commands in a supposedly secure Bluetooth connection. Moreover, the attacks are effective *regardless* of the victims' security capabilities, including Secure Connections and Numeric Comparison. We name our attacks *BLUR attacks*, as they are exploiting CTKD to blur the security boundary between BT and BLE.

We implement the BLUR attacks using a low-cost hardware and software setup that we will open-source. We demonstrate the effectiveness of the attacks by exploiting *16* unique devices employing *14* different Bluetooth chips from Broadcom, Cambridge Silicon Radio (CSR), Cypress, Intel, and Qualcomm. Our set of vulnerable devices covers Bluetooth 4.2, 5.0, 5.1, 5.2, which are the most popular in the market, and even a 4.1 device to which CTKD was backported. We could not test 5.3 devices because they were not in the market at the time of submission.

We concretely address the BLUR attacks by presenting four protocol-level countermeasures. Our mitigations can be implemented at the operating system level with low effort. We implemented a proof-of-concept of one of our proposed countermeasures (i.e., disable key overwriting) to protect a Linux laptop, and we successfully tested it against our attacks. We summarize our contributions as follows:

- We identify CTKD as a novel and relevant attack surface for Bluetooth. We find that CTKD enables to attack BT from BLE and vice versa (i.e., cross-transport attacks) while being standard-compliant, transparent to end-users and employed with BT and BLE most secure configurations.
- We design, implement, and evaluate four novel attacks targeting vulnerabilities in the specification of CTKD. The attacks are standard-compliant up to at least Bluetooth 5.0 (that should be the majority of CTKD-enabled devices according to [12, 15]). Our attacks are the first to exploit CTKD and act across BT and BLE transports compared to related work. They enable impersonation, MitM, and unwanted sessions attacks. We indicate them as BLUR attacks as they blur the security boundary between BT and BLE.
- We build a low-cost implementation of the BLUR attacks using a Linux laptop and a Bluetooth development board. We use our implementation to attack 16 devices employing 14 unique Bluetooth chips and covering all Bluetooth versions compatible with CTKD (e.g., 4.2, 5.0, 5.1, and 5.2). We demonstrate that the BLUR attacks are effective on all tested devices. Moreover, we discuss why the key overwrite countermeasure introduced since Bluetooth 5.1 is ineffective, and propose alternative countermeasures to mitigate the attacks.

Disclosure. We responsibly disclosed our findings with the Bluetooth SIG (which is supposed to contact the relevant vendors) two times. In May 2020, we sent our first report which is now tracked with CVE-2020-15802. In September 2020, the Bluetooth SIG unilaterally released a security note [13], claiming that Bluetooth 5.1 and later are not vulnerable to the presented attacks because of mitigations in the standard. We experimentally show that the mitigations are ineffective by exploiting 5.1 and 5.2 devices in Section 7.2, and discuss this further in Section 8. We disclosed our related findings to the SIG in May 2021.

2 BLUETOOTH CLASSIC (BT) AND LOW ENERGY (BLE)

BT and BLE are two wireless transports specified in the Bluetooth standard [14]. These transports complement each other: BT is used for high-throughput and connection-oriented services, such as streaming audio and voice, while BLE is optimized for very low-power and low-throughput services such as fitness tracking and digital contact tracing. High-end devices, such as laptops, smartphones, headsets, and tablets, provide both BT and BLE, while low-end devices such as mice, keyboards, and wearables provide either BT or BLE.

BT and BLE have similar security mechanisms (i.e., pairing and session establishment) but *different* security architectures and threat models. Pairing lets two devices establish and authenticate a pairing key that acts as the root of trust and is the standard is defined as secure simple pairing (SSP). BLE SSP is performed using the Security Manager Protocol (SMP) [14, p. 1597], while BT SSP uses the Link Manager Protocol (LMP) [14, p. 573]. During pairing, BLE allows negotiating the entropy of the pairing key while BT does not. BT and BLE provide a *Secure Connections* mode that enhances the pairing and session establishment security primitives. 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 [14, p. 266].

While pairing, BT and BLE employ similar *association mechanisms*. For example, if any of the two pairing devices have no input-output capabilities, then the devices use *Just Works (JW)* association. Just Works does not require user interaction, but it does not protect against MitM attacks. Session establishment lets paired devices establish a secure communication channel protected by a fresh session key derived from the pairing key. During session establishment, BT allows negotiating the entropy of the session key while the BLE session key inherits the entropy of the associated pairing key.

BT and BLE use the same notion of *pairable* and *discoverable* states. If a device is pairable, it accepts pairing requests from remote devices. If it is discoverable, it reveals its identity when scanned by other devices. Notably, a device answers a pairing request even if it is *not* discoverable [46]. For example, if the user knows the Bluetooth address of her pair of headphones, she can complete BT or BLE pairing by sending a pairing request from her laptop without putting the headphones into discoverable mode.

Both BT and BLE use a *Central-Peripheral* medium access protocol. The Central is the connection initiator, while the Peripheral is the responder. BT allows switching Central and Peripheral roles dynamically, while BLE roles are fixed. High-end devices, such as laptops and smartphones, support both BLE Central and BLE Peripheral modes and are typically used as BLE Centrals. Low-end devices, such as fitness trackers and smartwatches, support only the BLE Peripheral mode.

3 CROSS-TRANSPORT KEY DERIVATION

In this section, we introduce CTKD as a feature, and we describe its association protocols used from BT and BLE. In our descriptions, we refer to the Bluetooth Central as Alice and to the Peripheral as Bob and in the figure, we color-code BLE with light blue and BT with blue.

3.1 Introduction about CTKD

In 2014, Bluetooth 4.2 introduced CTKD to improve the *usability* of BT and BLE pairing. Before its introduction, devices had to pair over BT and BLE to use both of them securely. With CTKD, the devices pair either over BT or BLE, compute the pairing key, and derive the pairing key for the other transport without having to pair a second time [14, p. 276,1366].

A Bluetooth device requires few capabilities to support CTKD. It has to support BT and BLE (i.e., dual-mode), Secure Connections, and a Bluetooth version greater or equal to 4.2. Examples of devices supporting CTKD are laptops, tablets, smartphones, headsets, speakers, and high-end wearable devices, and their number is steadily growing [12]. The list of vendors includes Apple [48], Google [6], Cypress [18], Linux [16], Qualcomm [37], and Intel [26]. Notably, Apple presented it as a core and always-on Bluetooth feature during WWDC 2019.

The Bluetooth standard specifies a *custom* key derivation function (KDF) for CTKD [14, p. 1589]. The KDF takes a 128-bit key and two 4-byte strings and derives a 128-bit key. If CTKD starts from BLE, then the BT pairing key is derived using the “tmp2” and “brle” strings. In the other case, the derivation is performed using the “tmp1” and “lebr” strings. The standard uses KDF in a deterministic way as reusing the same input key will derive the same output key. We re-implemented KDF to validate our analysis (see Section A.1).

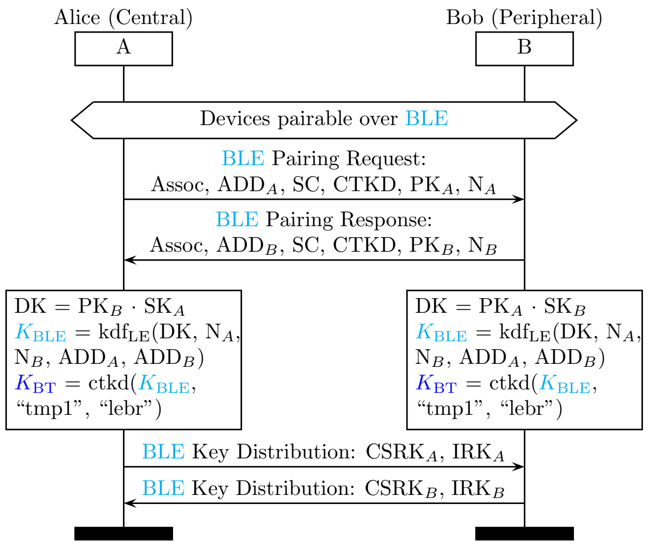
Alice (Central)
A
Bob (Peripheral)
B
Devices pairable over BLE
BLE Pairing Request: Assoc, ADD_A, SC, CTKD, PK_A, N_A
BLE Pairing Response: Assoc, ADD_B, SC, CTKD, PK_B, N_B
$DK = PK_B \cdot SK_A$
$K_{BLE} = kdf_{LE}(DK, N_A, N_B, ADD_A, ADD_B)$
$K_{BT} = ctkd(K_{BLE}, “tmp1”, “lebr”)$
$DK = PK_A \cdot SK_B$
$K_{BLE} = kdf_{LE}(DK, N_A, N_B, ADD_A, ADD_B)$
$K_{BT} = ctkd(K_{BLE}, “tmp1”, “lebr”)$
BLE Key Distribution: $CSRK_A$, IRK_A
BLE Key Distribution: $CSRK_B$, IRK_B

Figure 1: CTKD from BLE. Alice and Bob negotiate SC and CTKD support during BLE pairing. Then, they compute the BLE pairing key and derive the BT pairing key via CTKD. Note that they exchange no message over BT. Then, they exchange additional BLE keys, including signature (CSRK) and identity resolving (IRK) keys. After pairing with CTKD, the devices can establish secure sessions over BT and BLE.

Since CTKD is a novel attack surface, its security implications are unclear. Prior research did not cover CTKD, while the Bluetooth standard since version 5.1 provides limited CTKD security arguments. In particular, it only discusses mitigations against a particular class of key overwrite attacks (e.g., key overwrite with a weaker key). However, it ignores other threats such as other types of key overwrite attacks or attacks not requiring to overwrite keys.

3.2 CTKD Protocols for BT and BLE

In this section we describe how CTKD is negotiated and used from BLE and BT.

CTKD from BLE. Figure 1 shows how CTKD is negotiated and used to derive a BT pairing key during BLE pairing. Alice and Bob are pairable over BLE and BT and discover each other using BLE scanning and advertising. Then, they perform pairing over BLE using the SMP protocol. We experimentally found that CTKD is negotiated by setting to one the Link Key flag of the Initiator and Responder key distribution SMP fields [14, p. 1610] and that such negotiation is not protected. Other than the Link Key flag, the devices should also declare Secure Connections support (SC), which is also spoofable. The BLE pairing messages also contain an association method (Assoc), a source BLE address (ADD), a public key (PK), and a nonce (N).

After exchanging the pairing messages, the devices compute a Diffie-Hellman shared secret (DK) using the exchanged PK. DK is used to compute the BLE pairing key (K_{BLE}) using the BLE pairing key derivation function (kdf_{LE}). Then, the devices use CTKD's key derivation function (ctkd) to derive the BT pairing key (K_{BT}). To

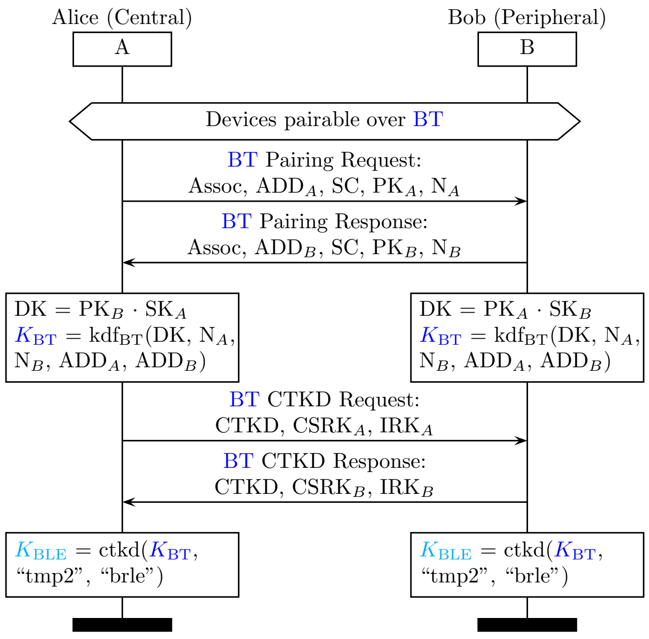
Alice (Central)
A
Bob (Peripheral)
B
Devices pairable over BT
BT Pairing Request: Assoc, ADD_A, SC, PK_A, N_A
BT Pairing Response: Assoc, ADD_B, SC, PK_B, N_B
$DK = PK_B \cdot SK_A$
$K_{BT} = kdf_{BT}(DK, N_A, N_B, ADD_A, ADD_B)$
$DK = PK_A \cdot SK_B$
$K_{BT} = kdf_{BT}(DK, N_A, N_B, ADD_A, ADD_B)$
BT CTKD Request: CTKD, $CSRK_A$, IRK_A
BT CTKD Response: CTKD, $CSRK_B$, IRK_B
$K_{BLE} = ctkd(K_{BT}, \text{"tmp2"}, \text{"brle"})$
$K_{BLE} = ctkd(K_{BT}, \text{"tmp2"}, \text{"brle"})$

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

complete BLE pairing, Alice and Bob establish a secure session over BLE and exchange additional keys (e.g., CSRK and IRK). As a result, Alice and Bob share K_{BLE} and K_{BT}, without having to pair over BT.

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

Unlike for BLE, BT pairing messages do not include a CTKD flag. Instead the devices start a secure BT session and exchange two messages containing the CTKD flag and additional security material needed for BLE, such as signature keys (CSRK) and identity resolving keys (IRK). These two messages are peculiarly encoded as BLE SMP packets but sent over BT. We are not sure why the Bluetooth standard is not describing such a "BLE tunneling" protocol. Once CTKD is negotiated, Alice and Bob use it to derive the BLE pairing key (K_{BLE}) from the BT key without pairing over BLE.

4 THREAT MODEL

We now present our system and attacker models.

4.1 System Model

Our system model considers two victims, Alice and Bob, who can securely communicate over BT and BLE. The victims support CTKD, use the most secure Bluetooth modes they support (e.g., SC and SSP with strong association), and are already paired over BT and/or BLE. This setup *should* protect the victims against eavesdropping, impersonation, and MitM attacks, as claimed in [14, p. 266]. Without loss of generality, we assume that Alice is the Central and Bob is the Peripheral.

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

4.2 Attacker Model and Goals

Our attacker model considers Charlie, an adversary in Bluetooth range with the victims. The adversary's knowledge is limited to what the victims advertise over the air, e.g., full or partial Bluetooth addresses, Bluetooth names, and security and IO capabilities. She can scan and discover devices, send pairing requests and responses, use CTKD, propose weak association mechanisms (e.g., Just Works), and dissect and craft Bluetooth packets.

The attacker does not know the secret pairing and session keys shared between the victims. She wants to conduct the attack at any time (e.g., she does not have to be present when the victims are pairing or negotiating a secure session). Moreover, she cannot access and tamper with the victims' devices.

The attacker has four goals. (i) *impersonate Alice (Central)* and take over (i.e., steals) her secure sessions with Bob. (ii) *impersonate Bob (Peripheral)* and take over his secure sessions with Alice. Central and Peripheral impersonation are different goals, as they require different attack strategies. (iii) *MitM* Alice and Bob's secure session (iv) establish *unintended sessions* with Alice and Bob as an anonymous device with arbitrary capabilities.

5 BLUR ATTACKS

In this section we introduce the BLUR attacks, four novel and high-impact attacks on CTKD. The attack enables to impersonate, MitM, and establish unintended session with any Bluetooth device supporting CTKD. We now describe the attacks' root causes, underlying attack strategy and technical details. We also discuss some interesting aspects about them.

5.1 Root Causes

Our attacks leverage two vulnerabilities associated with the design of CTKD:

Permanent BT and BLE pairing. CTKD incentivize vendors to keep devices pairable over BT and BLE to provide a better pairing experience (i.e., pairing once other than two times). However, this

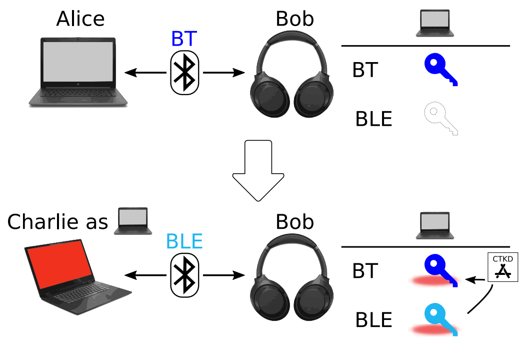

Figure 3: BLUR attack strategy. Alice and Bob are paired over BT and run a secure BT session. Charlie approaches Bob as Alice and starts a BLE process with Bob declaring CTKD support. Then Charlie agrees upon a BLE pairing key with Bob, and tricks Bob into overwriting Alice's BT and BLE pairing keys. As a result, Charlie can establish BT and BLE sessions with Bob as Alice.

can be exploited to pair on transports not currently in use, even if the victim is not discoverable on that transport.

Cross-transport key tampering. By design CTKD allows to write and overwrite pairing keys on BT and BLE by trusting a pairing process happening either on BT or BLE. As a result, CTKD exposes a novel cross-transport attack surface and associated attack vectors, where an attacker can try to exploit BT and BLE by targeting one of the two. For example, we abuse CTKD to tamper with pairing keys across transport, which is unprecedented for Bluetooth. Moreover, we attack CTKD to get access to BLE key material such as IRK and CSRK by only sending malicious BT packets.

Root causes impact. The attacks' root causes affect any Bluetooth device supporting CTKD, including, among others, IT devices (e.g., laptops, smartphones, and tablets) and IoT ones (e.g., headphones, earbuds, and speakers). Moreover, these vulnerabilities are present regardless of the Bluetooth application layer service (i.e., Bluetooth profile) as they target a link-layer security protocol.

5.2 Attack Strategy

Before explaining the BLUR attacks in detail, we present the attack strategy with the help of Figure 3. We assume an attack scenario where Alice is a laptop and Bob is a pair of headphones. The victims had already paired using CTKD. We find that this scenario can be exploited in multiple ways, we now describe the strategy needed to impersonate Alice to Bob. A similar strategy can be used to impersonate Bob to Alice.

As shown in the bottom part of Figure 3, Charlie can approach Bob as Alice and start a pairing process over BLE while declaring CTKD support. Bob completes pairing over BLE with Charlie thinking it is repairing with Alice. Then, Bob computes new BLE and BT pairing keys, overwrites Alice's keys, and connects with Charlie over BLE. As a result Charlie takes over Alice achieving his goal.

If Bob (or Alice) support strong pairing association, like Numeric Comparison, Charlie can optimize his attack strategy by *downgrading* it to a weak one (i.e., Just Works). In particular, Charlie can declare no input output capabilities during the BT/BLE pairing feature negotiation phase and trigger Just Works [25]. We note that this trick does *not* unset the pairing MitM protection flag.

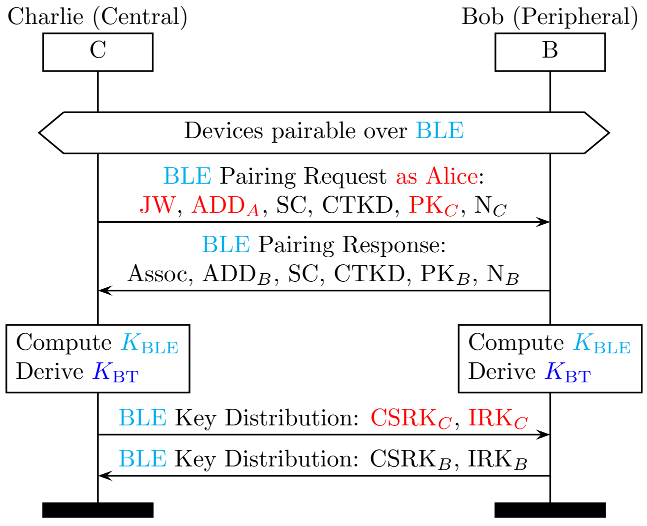
Charlie (Central)
Bob (Peripheral)
C
B
Devices pairable over BLE
BLE Pairing Request as Alice:
JW, ADD_A, SC, CTKD, PK_C, N_C
BLE Pairing Response:
Assoc, ADD_B, SC, CTKD, PK_B, N_B
Compute K_{BLE}
Derive K_{BT}
Compute K_{BLE}
Derive K_{BT}
BLE Key Distribution: $CSRK_C$, IRK_C
BLE Key Distribution: $CSRK_B$, IRK_B

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

5.3 Impersonation and MitM Attacks

We now describe the Central impersonation, Peripheral impersonation, and MitM BLUR attacks.

Central impersonation. Charlie impersonates Alice and takes over her BT and BLE sessions with Bob as in Figure 4. Charlie (Central) presents to Bob (Peripheral) as Alice and sends a BLE pairing request containing Alice's Bluetooth address (ADD_A), no input/output capabilities to trigger Just Works, his public key (PK_C), and CTKD support. Bob answers with a pairing BLE response believing that Alice wants to re-pair. Then, Charlie and Bob compute K_{BLE}, complete JW association, and derive K_{BT} via CTKD. Bob ends up overwriting Alice's BT and BLE pairing keys with the newly derived keys. Additionally, Charlie and Bob exchange additional BLE key material (e.g., CSRK, IRK).

Peripheral impersonation. Charlie impersonates Bob and takes over his BT and BLE sessions with Alice as in Figure 5. In this case, Charlie (Peripheral) impersonates Bob to Alice and sends her a BT pairing request containing Bob's address (ADD_B), no input/output capabilities, and his public key (PK_C). Even if Charlie is a Peripheral, she can send a BT pairing request because BT allows switching roles before starting a pairing process. Alice answers with a BT

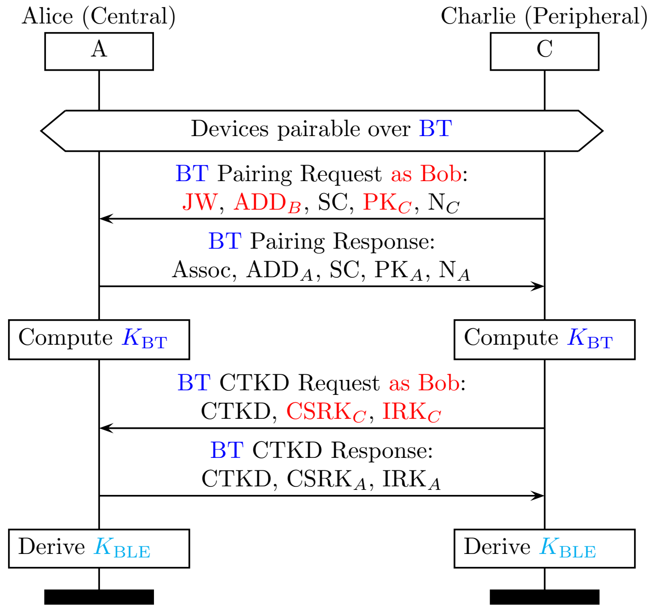

Figure 5: BLUR Peripheral impersonation attack. Charlie sends a BT pairing request with Bob's address (ADD_B) including Just Works association, and his public key (PK_C). The pairing request is valid as BT enables to dynamically switch from Peripheral to Central before sending a pairing request. Alice 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 Peripheral impersonation attack, Charlie tricks Alice into overwriting Bob's keys with his ones and takes over Bob who can no longer connect back to Alice.

pairing response believing that Bob wants to re-pair over BT. The two compute K_{BT} with JW association.

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 Charlie's signature and MAC randomization BLE keys ($CSRK_C$, IRK_C) and the CTKD support flag. Alice answers with a BLE pairing response tunneled over BT. Then the devices derive K_{BLE} via CTKD. As a result, Charlie forces Alice into overwriting Bob's pairing keys with his keys and accesses additional BLE keys from Alice.

MitM. Charlie mounts a MitM attack by combining the Central and Peripheral spoofing attacks as shown in Figure 6. If Alice and Bob are running a BLE session, Charlie starts with the Peripheral impersonation attack presenting to Alice as Bob over BT. Otherwise, he launches a Central impersonation attack by targeting Bob as Alice over BLE. After the first attack, the impersonated victim is 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 sits

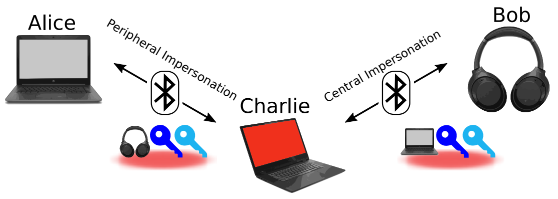

Figure 6: BLUR MitM attack. Charlie combines the Central and Peripheral impersonation attacks and MitM Alice and Bob secure session over BT and BLE.

in the middle of secure BT and BLE sessions between Alice and Bob.

5.4 Unintended Session Attacks

During our experiments with CTKD, we noticed that all devices were always pairable over BT and BLE even when they were not discoverable. However, while being pairable over BT and BLE devices typically use one transport at a time. For example, a pair of headsets advertises its presence over BT and BLE, but once paired uses only BT audio profiles. The attacker can target the unused transport to establish secure BT and BLE session with a victim while impersonating a random device. We name this novel Bluetooth threat as *unintended session attack*.

An unintended session attack is valuable for various reasons. It is *stealthy* as the attacker pairs with a victim as an anonymous device and with minimal user interaction. Moreover, it allows *complete device enumeration* as the attacker, being a trusted peer, can access all BT and BLE services, including protected ones (unlike related attacks [17]). Additionally, the attack anonymously *de-anonymizes* victim devices, as the attacker gets access to the identity resolving key distributed during pairing. Finally, the trust relation between the attacker and the victim enables the adversary to reach *more* Bluetooth code sections, including potential remote code execution bugs in the pairing and secure session code.

In Figure 7, we describe an unintended session attack against Bob, who is securely connected to Alice over BT. Charlie can send a BLE pairing request to Bob with a random Bluetooth address, CTKD support, and Just Works association. Charlie and Bob negotiate K_{BLE}, and derive K_{BT} via CTKD. Now, Charlie can establish secure but unintended BT and BLE sessions with Bob without breaking Bob's existing session with Alice. Charlie can also establish unintended sessions with Alice using a similar strategy.

6 IMPLEMENTATION

In this section, we describe our attack scenario, and our implementation of a custom attack device to perform the BLUR attacks that we will open-source.

6.1 Attack Scenario

Figure 8 presents our attack scenario. 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

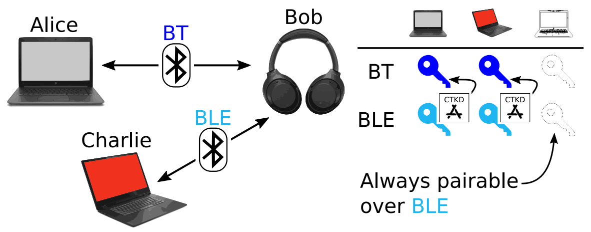

Figure 7: BLUR unintended sessions attack. Charlie can take advantage of CTKD to establish unintended BT and BLE sessions with Bob as a random device with arbitrary capabilities. The same can happen if Charlie targets Alice.

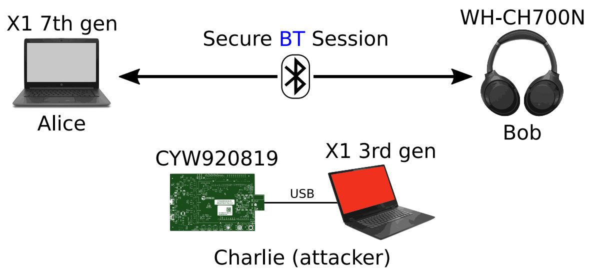

Figure 8: BLUR attack scenario. Alice (Central) is a ThinkPad X1 7th gen, Bob (Peripheral) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in the absence of Charlie, and are running a secure BT session.

development board [19] and a 3rd generation ThinkPad X1 laptop as her attack device. See Section 6.2 for the attack device's implementation.

Table 1 summarizes the most relevant Bluetooth 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 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.

6.2 Attack Device

Our attack device consists of a *CYW920819 development board* and a *Linux laptop* (see Figure 9) and is convenient for several reasons. We can program our board (Bluetooth Controller) to impersonate any BT/BLE device, we can patch its closed-source firmware to control the BT link management protocol (LMP) packets and the BLE link layer ones. 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 discarded a software-defined radio (SDR) setup as there is no fully functional and open-source BT/BLE SDR stack. We now describe in detail how we modify the attack device's Host and Controller components.

Table 1: Relevant features of Alice, Bob, and Charlie. We redact the devices' Bluetooth addresses for privacy reasons.

	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	Central	Peripheral	Central
IO	Display	No IO	Display
Association	Numeric C.	Just Works	Numeric C.
Pairable	True	True	True

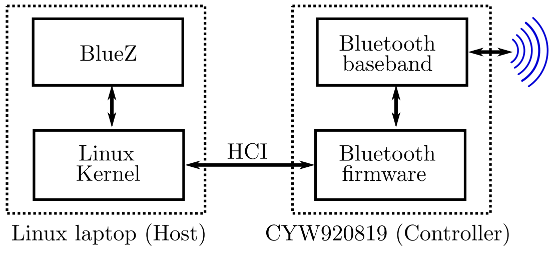

Figure 9: Attack device block diagram. The attack device is composed of Linux laptop (Host) and a CYW920819 development board (Controller) connected via USB and communicating using the Host Controller Interface (HCI) protocol.

Host modifications. We modified the host using standard Linux tools such as `bluetoothctl`, `hciconfig`, and `btmgmt`. For example, we use `btmgmt` to toggle several Bluetooth functionalities (e.g., BT, BLE, SC, scanning, advertising, and discoverability) and to send pairing requests with no input-output capabilities.

Furthermore, we patch the host to parse the link-layer packets from the controller. This is handy as it enables monitoring Bluetooth HCI and link-layer traffic solely from the host. To activate link-layer packet forwarding on the controller, we sent a vendor-specific command to the board that switches on an undocumented diagnostic mode. Finally, we patched the host such that the MitM flag is always set to True, even when the input-output capability flag of the attack device is set to None.

Controller modifications. We modified the controller by dynamically patching the development board Bluetooth firmware. The patches are sent using a proprietary mechanism supported by the board. To develop the patches, we had first to dump the firmware, and reverse-engineer its relevant parts. Using a proprietary HCI command we dumped a board RAM snapshot at runtime. We use the memory maps that we extracted from the board's SDK to extract the memory segments from the snapshot (e.g., ROM, RAM, and the scratchpad). The firmware is stored in the ROM as a stripped ARM32 binary and contains 16-bit Thumb instructions.

To reverse-engineer the firmware, we loaded the ROM, RAM, and scratchpad in Ghidra and statically analyzed them. In our first pass, we isolated the libc functions (e.g., `malloc` and `calloc`) by looking at the signatures and code patterns of the most called functions. Then, as described in [33], we loaded into Ghidra the firmware debugging symbols from the board's SDK, and we isolated interesting functions and data structures. Then, we wrote ARM Thumb assembly patches to change their behaviors, and we applied those patches at runtime using internalblue, a toolkit presented in [33]. Our set of patches allows transforming our board into whatever device we want by changing its identifiers, including addresses, names, and capabilities.

7 EVALUATION

We now present our evaluation setup and results.

7.1 Setup

We now describe how we conducted the four BLUR attacks presented in Section 5 using the attack device described in Section 6.2.

Central impersonation. To impersonate a Central (e.g., a laptop), we patch our attack device to clone the Central Bluetooth features (e.g., Bluetooth address, name, device class, and security parameters). To start the attack, we send a BLE pairing request declaring CTKD and Just Works support using `btmgmt`. The devices agree on K_{BLE}, derive K_{BT} via CTKD and establish a secure BLE session. Then, the Peripheral terminates the BT session with the impersonated Central, and starts a secure BT session with the attack device.

Peripheral impersonation. To impersonate a Peripheral (e.g., a pair of headphones), we modify our attack device to mimic the Peripheral's Bluetooth features. Then, we send a BT pairing request from the attack device to the victim Central declaring CTKD and Just Works support using `btmgmt`. The Central and the attack device agree on K_{BT}, derive K_{BLE}via CTKD, and establish a secure session over BT.

MitM. By using two development boards connected to the same laptop, we impersonate the Central and the Peripheral at the same time to mount a MitM attack. In particular, we run the Central impersonation attack first, and then the Peripheral impersonation one.

Unintended sessions. To conduct the unintended session attack, we patch our attack device to spoof a device unknown to the victim. If we target a Central (Peripheral) the attack starts with a pairing request over BT (BLE). In both cases, the attacker completes pairing using CTKD and can establish secure sessions over BT and BLE with the victim as an anonymous device.

7.2 Results

Table 3 presents the evaluation results. The first three columns indicate the device producer, model, and operating system (OS). The following two columns provide information about the Bluetooth chip producer and model. The next column shows the Bluetooth version. The last four columns indicate the role of the attacker (e.g., if the adversary is a Peripheral then the victim is a Central), and whether or not a victim is vulnerable to impersonation, MitM, and unintended sessions (US) attacks. If a device is vulnerable, we mark the table cell with a checkmark (✓).

Overall we exploit 16 unique devices employing 14 different Bluetooth chips. We targeted different device types (e.g., laptops, smartphones, headphones, and development boards), manufacturers (e.g., Samsung, Dell, Google, Lenovo, and Sony), operating systems (e.g., Android, Windows, Linux, and proprietary OSes), and Bluetooth firmware (e.g., Broadcom, CSR, Cypress, Intel, Qualcomm, and Samsung).

Moreover, Table 3 empirically demonstrates that the BLUR attacks are effective on all the Bluetooth versions that we tested (Bluetooth 4.2, 5.0, 5.1, and 5.2). We could not test Bluetooth 5.3 devices as they were not available at the time of submission. Notably, the attacks work also against Bluetooth 4.1 devices.

Finally, Table 3 shows that the Bluetooth 5.1 and 5.2 devices that we tested are vulnerable the BLUR attacks, despite the CTKD key overwrite mitigation in the standard since Bluetooth 5.1 [14, p. 1366]. In Section 8, we discuss why we think that this is the case and we propose alternative and effective countermeasures.

8 COUNTERMEASURES

In this section, we summarize the CTKD key overwrite countermeasures in the Bluetooth standard since version 5.1, and we explain why they are not effective against the BLUR attacks. Then, we propose effective mitigations for the BLUR attacks, and we describe how we implemented and evaluated one of them.

8.1 Countermeasures in the Bluetooth standard and their ineffectiveness

Since 5.1 the Bluetooth standard introduced backward incompatible countermeasures against some CTKD key overwrite attacks. In particular, the 5.1 and 5.2 standards state "While performing cross-transport key derivation, if the key for the other transport already exists, then the devices shall not overwrite that existing key with a key that is weaker in either strength or MITM protection" [10, p. 1401]. Hence, an attacker cannot overwrite a pairing key with CTKD if the overwriting key has either a lower entropy (i.e., strength) or a lower MitM protection than the overwritten one. The phrasing is 5.3 is more verbose, but it enforces the same conditions [14, p. 1366].

The Bluetooth SIG currently relies on these countermeasures to claim that the BLUR attacks are ineffective against devices supporting Bluetooth 5.1 and later [13]. Nevertheless, the results presented in Section 7.2 demonstrate that such mitigations appear to be ineffective against our attacks. The interesting question then is: "Why are the BLUR attacks successful against Bluetooth 5.1 or higher?" One might expect implementation issues, but we argue that the problems still lie with the design of CTKD for the following three reasons:

- None of our attacks use CTKD to downgrade the pairing key's strength (i.e., entropy). Indeed, they do not trigger the key strength condition of the countermeasure.
- The unintended session attacks never require overwriting pairing keys via CTKD. The same is true for impersonation attacks where the adversary attacks a transport and the victims are only sharing a pairing key for that transport as the pairing key derived from CTKD does not overwrite an old key.
- Some impersonation attacks require overwriting keys via CTKD. However, we experimentally found that for 5.1 and 5.2 devices, if we declare no input/output capabilities without unsetting the MitM protection flag, then we are still capable of downgrading pairing to Just Works without triggering the mitigation in the Bluetooth standard. This is a consequence of the association negotiation criteria in the Bluetooth standard [14, p. 1573].

8.2 Our Countermeasures

We now propose two countermeasures against our attacks filling the security gaps still present in the Bluetooth standard. Our countermeasures directly address the permanent pairing and key tampering root causes described in Section 5.1 and prevent the four BLUR attacks.

Avoid cross-transport key tampering. To address CTKD key tampering problems (including key overwrites), we suggest disabling CTKD when is used to overwrite trusted pairing key, unless the user explicitly consents (with sufficient explanations to make an informed decision). This countermeasure should be added to any device supporting CTKD, not only from standard version 5.1 onwards.

Disable permanent BT and BLE pairing. To address the issues with devices being permanently pairable over BT and BLE, we recommend that a device should automatically stop being pairable on a transport that is not currently in use. To avoid denial of service issues a device should also allow to manually enable and disable pairing on BLE and BT.

8.3 Key Tampering Countermeasure PoC

To verify the effectiveness of the cross-transport key tampering countermeasure, we developed and successfully tested a proof of concept (PoC) key tampering mitigation for Linux. We selected Linux because is open-source and is employed on multiple Bluetooth devices such as Android smartphones, embedded devices, and laptops.

Our PoC works as follows. We pair a Linux laptop (victim) with a smartphone using CTKD. Let us assume that the laptop Bluetooth address is `AA:AA:AA:AA:AA:AA` and the smartphone one is `BB:BB:BB:BB:BB:BB`. Once the pairing process is completed, the laptop stores the pairing keys in the `info` file at: `/var/lib/bluetooth/AA:AA:AA:AA:AA:AA/BB:BB:BB:BB:BB:BB/info`. To prevent key tampering, we unset the write permission bit on such `info` file.

Table 2: Comparison with related work. The BLUR attacks are the first *cross-transport* standard-compliant attacks for Bluetooth and the first targeting *CTKD*. C = Data Confidentiality, I = Data Integrity, A = Device Authentication, K = Key disclosure. No (○) Partially (◐), Yes (●).

			Attack			
Year	Paper	Target	Phase	C I A K	SC	Pst
Attacks on BT						
2016	Albaz. [1]	Standard	Any	◐○○○	x	-
2017	Seri [40]	Impl.	Pairing	●●●○	NA	✓
2018	Sun [43]	Standard	Pairing	●●●○	✓	-
2018	Biham [9]	Standard	Pairing	●●●◐	NA	✓
2019	Ossmann [35]	Standard	NA	◐○○○	x	-
2019	Antonioli [2]	Standard	Pairing	●●◐○	✓	-
2020	Antonioli [4]	Standard	Pairing	●●●○	✓	-
2021	Tsch. [44]	Standard	Pairing	●●●○	✓	-
Attacks on BLE						
2016	Jasek [28]	Standard	NA	◐○○○	x	-
2018	Biham [9]	Standard	Pairing	●●●◐	NA	✓
2019	Seri [41]	Impl.	NA	○◐◐○	NA	✓
2020	Zhang [49]	Standard	Pairing	◐◐◐○	✓	-
2020	Wu [47]	Standard	Session	○○●○	✓	-
2020	Garbelini [22]	Impl.	Any	◐◐◐○	NA	-
2020	Antonioli [5]	Standard	Pairing	●●◐○	✓	-
Cross-transport attacks on BT and BLE						
2022	**BLURtooth**	Standard	Any	●●●◐	✓	✓

Then, we run the BLUR Peripheral impersonation attack against the laptop (Central) and we confirm that the attack is ineffective.

Our PoC can be improved by selectively setting and unsetting the write permission on the pairing key file based on manual or automatic triggers. For example, the OS might integrate a graphical user interface to lock/unlock the file. Moreover, it can provide an automatic mechanism that locks the pairing key file if a trusted device tries to repair with weaker Bluetooth security parameters (e.g., Just Works other than Numeric Comparison). We note that effectively detecting impersonation attacks is outside of the scope of this countermeasure.

9 RELATED WORK

Bluetooth is the most popular wireless technology for low-power services and cable replacement [23]. The Bluetooth standard evolved to include more security mechanisms (e.g., SSP, SC) and transports (e.g., BT, BLE). Over the years, numerous standard-compliant attacks have been discovered since Bluetooth v1.0 [27, 32].

Recent standard-compliant attacks on BT include attacks on legacy pairing [42], secure simple pairing (SSP) [9, 24, 43], association [25, 44], key negotiation [2], and authentication procedures [4, 31, 45]. Regarding BLE, there exist attacks on legacy pairing [39], key negotiation [5], SSP [9, 49], reconnections [47], and GATT [28].

The BLUR attacks are novel compared to *prior* standard-compliant attacks. As we can see from Table 2, they are the first *cross-transport* attacks, meaning the first targeting BT from BLE and vice versa. Moreover, no prior attacks targeted (and evaluated the security of) CTKD. Finally, like the BIAS attack [4], they require a weak threat model as the attacker can target a victim at any time. Unlike the BIAS attack, the effect of our attacks is persistent across sessions. Like other standard-compliant attacks, the BLUR attacks are effective regardless of the security mode (e.g., SSP with SC), association method (e.g., Numeric Comparison), and Bluetooth version numbers. Surveys on Bluetooth security are not discussing CTKD, which instead should be part of Bluetooth's threat model [20, 34, 36].

We have seen attacks targeting specific implementation flaws on BT [40] and BLE [22, 41]. As our attacks target the CTKD specification, they are effective regardless of the implementation details.

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

The cryptographic primitives used by Bluetooth have been extensively analyzed. For example, the E_0 cipher used by BT was investigated [21], and it is considered relatively weak [36]. SAFER+, used for authentication, was analyzed as well [30]. BT and BLE "Secure Connections" use the AES-CCM authenticated-encryption cipher. AES-CCM was extensively analyzed [29, 38], and it is FIPS-compliant. The BLUR attacks target the CTKD protocol, hence they are effective with perfectly secure cryptographic primitives.

10 CONCLUSION

CTKD was introduced in Bluetooth 4.2 to improve the usability of BT and BLE pairing. However, its security implications were not clear as there was no prior research evaluating its security. Moreover, the Bluetooth standard provides only limited and version-specific security arguments about CTKD since version 5.1. This work demonstrates that CTKD is a novel and relevant attack surface as it enables high impact attacks affecting BT and BLE just by targeting one of the two (i.e., unprecedented cross-transport exploitation for Bluetooth).

We present the BLUR attacks, four novel cross-transport threats on CTKD to impersonate, MitM and establish unintended (anonymous) sessions with BT and BLE device. Since the attacks target vulnerabilities in CTKD's specification, they are effective regardless of the security capabilities of the victim (e.g., usage of SSP, SC, on strong association) and software and hardware details. The attacks exploit the fact that CTKD, by design, allows an attacker to pair over unused transports and tamper with security keys across transports (e.g., write and overwrite pairing keys).

To demonstrate the feasibility of the BLUR attacks, we present a low-cost implementation based on readily available hardware and open-source software (that we will open-source). We use our implementation to confirm that the BLUR attacks are effective and impactful. In particular, we exploited 16 different devices using 14 unique Bluetooth chips. Our device sample includes all the Bluetooth version currently in the market supporting CTKD (e.g., 4.2, 5.0, 5.1, and 5.2) and devices supporting SSP, SC and strong association.

We also experimentally confirm that the mitigations in the Bluetooth standard since v5.1 are ineffective against the BLUR attacks, and we provide solid arguments to explain why. As such, we expect all current CTKD-capable Bluetooth devices to be vulnerable to the BLUR attacks. To fix the attacks, we propose two practical and legacy-compliant countermeasures. In particular, we suggest disabling pairing when not needed and adding a mechanism to prevent key tampering. We implemented the latter on Linux and successfully evaluated it against a BLUR impersonation attack.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers. This project was supported by European Research Council (ERC) grant No. 850868, DARPA HR001119S0089-AMP-FP-034, and Fondation Botnar. Any findings are those of the authors and do not necessarily reflect the views of our sponsors.

REFERENCES

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

[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation of Bluetooth BR/EDR. In *Proceedings of the USENIX Security Symposium*. USENIX.

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

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

[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *Transactions on Privacy and Security (TOPS)* (2020). https://doi.org/10.1145/3394497

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

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

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

[9] Eli Biham and Lior Neumann. 2018. Breaking the Bluetooth Pairing–Fixed Coordinate Invalid Curve Attack. http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf.

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

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

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

[13] Bluetooth SIG. 2020. Bluetooth SIG Statement Regarding BLURtooth. https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/blurtooth/.

[14] Bluetooth SIG. 2021. Bluetooth Core Specification v5.3. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=521059, Accessed: 2021-11-15.

[15] Bluetooth SIG. 2021. Bluetooth Market Update 2021. https://www.bluetooth.com/bluetooth-resources/2021-bmu/?utm_campaign=bmu&utm_source=internal&utm_medium=web&utm_content=2021bmu-resourcepopup.

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

[17] Guillaume Celosia and Mathieu Cunche. 2019. Fingerprinting Bluetooth Low Energy devices based on the generic attribute profile. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. 24–31.

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

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

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

A APPENDIX

A.1 Implementation of CTKD KDF

We implemented CTKD's KDF, following the Bluetooth standard [14, p. 1366]. This implementation is *not* required to conduct the attack, but it is useful to check that CTKD keys are correctly derived. Our implementation is written in Python 3, uses the PyCA cryptographic module [7], and successfully passes the test vectors in the Bluetooth standard [14, p. 1650].

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

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

KDF uses the function $f(a, b)$. This function is implemented as AES-CMAC$(key, plaintext)$ and depends on the negotiation of h7 via the AuthReq flag [14, p. 1567]. When CTKD is run from BT, then K_{BLE} is computed by calling f two times with different input. If the devices support the h7 key conversion function, then f is called once with the "tmp2" and K_{BT}, and then a second time using the output of the first call and "brle", Otherwise, if h7 is not supported in the first f call the inputs are swapped. A similar logic with different inputs is used when CTKD is run from BLE to compute K_{BT} (see equations above).

Table 3: BLUR attacks evaluation results. The first three columns show the device's producer, model, and OS. The following two columns state the Bluetooth chip's producer and model. The sixth column tells the Bluetooth version of the target device. The seventh column indicates the attacker's role. The last three columns contain a checkmark (✓) if a device is vulnerable to the relevant BLUR attack.

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

† CTKD was backported by the vendor to Bluetooth 4.1.