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

Anonymous Author(s)

ABSTRACT

The Bluetooth standard specifies two different Bluetooth protocols: Bluetooth BR/EDR or "classic" (BT) for high-throughput services such as audio and voice, and Bluetooth Low Energy (BLE) for very low-power services such as monitoring and localization. BT and BLE use similar security mechanisms such as pairing but have different security architectures and threat models. Since pairing the same devices twice is considered "user-unfriendly", Bluetooth v4.2 introduced Cross-Transport Key Derivation (CTKD). CTKD lets two devices pair once, either over BT or BLE, and generate key material to securely use both BT and BLE.

We present the first security analysis of CTKD and identify seven novel cross-transport vulnerabilities in CTKD. Based on these vulnerabilities, we design and implement four novel cross-transport attacks, enabling an attacker to impersonate devices, manipulate traffic between two victims, and to establish malicious sessions. As our attacks are standard-compliant, they are effective against any device supporting CTKD regardless of the implementation details. We name our attacks BLUR attacks, as they blur the security boundary between BT and BLE. We experimentally demonstrate the BLUR attacks on 13 devices from different providers using 10 unique Bluetooth chips, and discuss effective countermeasures for the BLUR attacks. We have disclosed our findings and our 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 that is maintained by the Bluetooth SIG and we base our research on the current version of the standard, Bluetooth v5.2 [12]. The standard specifies two wireless stacks, Bluetooth Classic (BT) and Bluetooth Low Energy (BLE). BT and BLE are different and incompatible wireless stacks. BT is best suited for high-throughput use cases, such as streaming audio, voice calls, and voice control, while BLE is best suited for very low-power use cases such as localization and monitoring.

The standard defines *separate* security architectures and threat models for BT [12, p. 947] and BLE [12, p. 1617]. While these architectures address different threat models, they do so using similar security mechanisms, including *pairing* and *secure session establishment*. Pairing enables devices to establish a long term key, and secure session establishment enables paired devices to establish a secure session protected by a session key derived from the long term key.

Two devices who support both BT and BLE have to pair two times to use both transports securely. As pairing the same devices twice is considered "user-unfriendly", Bluetooth v4.2 introduced *Cross-Transport Key Derivation (CTKD)*. CTKD enables devices to pair once, either over BT or BLE, and generate both BT and BLE long term keys without having to pair a second time [12, p. 1401]. All the major Bluetooth software stacks (Apple, Linux, Android, and Windows) and hardware providers (Cypress, Intel, Qualcomm, Broadcom, Apple, Sony, and Bose) implement CTKD. For example, Apple at WWDC 2019 presented CTKD as a core Bluetooth feature that is enabled by default to improve BT and BLE usability [37].

CTKD is a rich 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 on both transports). Despite this fact, the security implications of CTKD are not understood and CTKD remains an unexplored attack surface. For example, the standard does not include CTKD in the BT and BLE threat models and we are not aware of any public security analyses of CTKD.

We present the first security analysis of CTKD (see Section 3), uncovering seven novel vulnerabilities in CTKD. Those vulnerabilities enable novel cross-transport attacks, that are capable of exploiting both transports just by targeting one of them. In particular, we design and implement four cross-transport attacks to allow impersonation, interception, and manipulation of traffic between victims, as well as unintended device sessions. Our attacks are standard-compliant and are thus effective against any device which supports CTKD. Our attacks complement the state-of-the-art such as [2–4, 10, 20, 21, 31, 34], as they are the first to exploit CTKD and enable BT and BLE cross-transport exploitation (detailed in Section 4). We call our attacks *BLUR* attacks, as they blur the security boundary between BT and BLE.

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 affordable and easy. Our evaluation (Section 5) demonstrates that all the tested devices are vulnerable. We will release our tools to the public after the responsible disclosure procedure is complete. We use our attack implementation to evaluate 13 devices, with 10 unique Bluetooth chips, from most of the major hardware and software vendors, e.g., Apple, Broadcom, Cambridge Silicon Radio (CSR), Cypress, Google, Intel, Linux, Qualcomm, and Windows and representing all Bluetooth versions that support CTKD (i.e., 4.1, 4.2, 5.0, and 5.1).

We summarize our contributions as follows:

- We perform the first security analysis of CTKD, a feature that enables to cross the security boundary between BT and BLE. We identify seven novel and very serious vulnerabilities, which enable the first cross-transport attacks between BT and BLE.
- We describe how to exploit the vulnerabilities in CTKD. We design and implement man-in-the-middle, impersonation, and unintended session attacks. We collectively call these BLUR attacks, as they blur the security boundary between

BT and BLE. We present a low-cost implementation for the BLUR 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 attack tools on 13 unique devices. We provide mitigation strategies to address the attacks directly in the Bluetooth standard. We have disclosed our findings and mitigations to the Bluetooth SIG in May 2020.

2 BACKGROUND

Here, we compare BT and BLE, and we introduce CTKD.

2.1 A Comparison of BT and BLE

BT and BLE are two wireless technologies specified in the Bluetooth standard. Such technologies are incompatible (e.g., same 2.4 GHz band but different physical layers and link layers) and are designed to complement each other. BT is used for high-throughput and connection-oriented services, such as streaming audio and voice. BLE is used for very low-power and low-throughput services such as localization and monitoring. Typically, high-end Bluetooth devices provide both stacks in single radio chip.

BT and BLE have similar security mechanisms but different security architectures and threat models. Both stacks provide a pairing mechanism (SSP) to let two devices establish a long term key. BLE allows negotiating the entropy of the long term key while BT does not. Both stacks provide a secure session establishment mechanism to derive a session key from the long term key and protect the communication. 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.

Both stacks support a “Secure Connections” mode that uses FIPS compliant security primitives such as AES-CCM for authenticated encryption, ECDH on 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 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 protection and Numeric Comparison where the user has to confirm to see the same numeric code on the pairing devices.

BT and BLE define master and slave roles, however they use them differently. For BT, the master is the connection initiator and the slave is the connection responder, and either the master or the slave can request a role switch. For BLE master and slave roles are fixed and switching roles is not supported. The master acts as a server (known as the BLE central) and the slave acts as a client (known as the 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 and smartwatches, implement only the slave mode.

All the presented differences are a consequence of using similar mechanisms on different security architectures and the result is that *BT and BLE have different threat models*. For example, attacking BLE pairing would allow reducing the entropy of the long term key and the related session keys while attacking BT pairing would not allow reducing the entropy of the long term key.

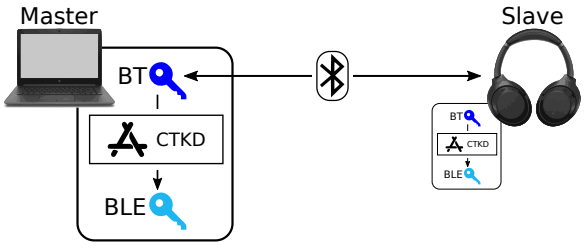

Figure 1: CTKD overview. After pairing and exchange a pairing key over BT CTKD is used to derive another pairing key for use with BLE. It is also possible to do the initial pairing over BLE, and use CTKD to generate the BT pairing key.

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 and this is not user-friendly. The Bluetooth standard addressed this issue in Bluetooth 4.2 by introducing CTKD. 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).

The Bluetooth standard specifies the same CTKD function for BT and BLE. This function takes as inputs a 16-byte key and two 4-byte strings and derives a 16-byte key. For BT, the CTKD function derives a BLE long term key (K_{BLE}) from a BT long term key (K_{BT}) and the strings "tmp2" and "brle". In case of BLE, the CTKD function derives K_{BT} from K_{BLE} and the strings "tmp1" and "lebr". Note that those are the actual keys defined by the standard and the keys used in all implementations we analyzed. As the standard defines all the 4-byte strings, CTKD derives the same output key when reusing the same input key. CTKD internally uses a sequence of AES-CMAC, see Section 5.3 for more details.

CTKD is widely supported and used by vendors such as Apple [37], Google [5], Cypress [15], Linux [14], Qualcomm [30], and Intel [22]. CTKD is used in combination 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 AES-CCM authenticated-encryption for BT, and ECDH pairing for BLE.

3 SECURITY ANALYSIS OF CTKD

As introduced in Section 2.2 and depicted in Figure 1, CTKD improves the usability of Bluetooth 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. CTKD is an interesting attack target as it crosses the security boundary between BT and BLE, and is used in combination with Secure Connections (the most secure mode for Bluetooth). Despite that, the Bluetooth standard does not provide a security evaluation of CTKD and does not include it in the BT and BLE threat models [12, p. 1401].

We present the first security analysis of CTKD based on the information available in the Bluetooth standard and our experiments. In total, we present seven novel cross-transport vulnerabilities that

we isolate in CTKD as a result of our analysis. The uncovered vulnerabilities enable to exploit BT and BLE just by targeting one of them. Then we explain how such vulnerabilities can be used to violate fundamental security guarantees promised by the Bluetooth standard. In Section 4 we explain how to exploit the identified vulnerabilities.

3.1 Identified Vulnerabilities in CTKD

Cross-Transport Pairing. CTKD requires that a device is pairable over BT and BLE, and continues to be pairable over at least one transport over time. This does not happen with conventional BT and BLE pairing where a device enters pairing mode, pairs, and then exits pairing mode. This issue enables an attacker to pair with a victim device in situations where the victim devices should not be pairable. For example, an attacker can pair with a pair of headphones over BLE even if the headphones are already paired with a legitimate device and are running a secure session over BT.

Cross-Transport Key Overwrite. CTKD enables an attacker to (over)write a long term key for one transport by pairing on the other one. This issue enables an attacker to maliciously overwrite long term keys on BT and BLE by controlling either BT pairing or BLE pairing. For example, an invalid curve attack [10] on BT would enable the attacker to compute BT and BLE long term keys. The same holds for other types of attacks such as key reinstallations.

Cross-Transport Authentication. CTKD enables to authenticate a device on one transport and then extend the claim to the other transport. For example, an attacker who impersonates a device on one transport can extend the effect of the impersonation attack to the other transport and even take over an existing session between the impersonated device and another victim device. As a result of a takeover, the impersonated device cannot connect back to the victim as she does not own the new long term key established by the attacker and the victim.

Cross-Transport Association. CTKD enables to use an association mechanism on one transport that is weaker than the one used for the other transport. For example, an attacker can pair on one transport using Just Works to avoid user interaction and disable protection against a man-in-the-middle, even if the other transport expects to pair using an association mechanism that requires user interaction and protection against a man-in-the-middle.

Cross-Transport Roles. Both BT and BLE include the notion of master and slave roles, yet they define such roles differently. For BLE, one device is always the master (also known as the central) and the other is always the slave (also known as the peripheral). For BT instead, the master and the slave roles can be changed on demand. In both cases, the master sends pairing requests and the slave sends pairing responses. This issue enables the attacker to target a victim with unexpected but valid pairing requests or responses. For example, a BLE master victim might only expect pairing responses from a legitimate BLE slave but can be targeted with BT pairing request from a malicious BT master.

Cross-Transport Secure Connections. CTKD enables to attack devices that do not support Secure Connections on both transports. In our experiments, we find that we can pair over BLE using CTKD with headphones that only support BLE Secure Connections. This issue considerably increases the number of vulnerable devices, as an attacker is not limited to target only devices which support BLE and BT Secure Connections at the same time.

Cross-Transport Unintended Sessions. In some CTKD use cases, one transport is expected to be used only for pairing. For example, Bluetooth speakers and headsets enable to pair over BLE with CTKD and then expect to only use BT, leaving the BLE transport open to attacks. This issue enables an attacker to establish stealthy malicious sessions with the victim on the transport that is expected to be used only for pairing without affecting existing sessions. If the left-open transport is BLE, then the attacker can run multiple unintended sessions with the victim as BLE support parallel connections.

3.2 Security Expectations Violated by CTKD

CTKD enables crossing the security boundary between BT and BLE and opens new opportunities for cross-transport attacks. We now list three security expectations that we found to be violated as a consequence of the vulnerabilities introduced by CTKD.

Security Boundary. The Bluetooth standard should guarantee that an attacker cannot attack one transport by exploiting weaknesses on the other transport. The BT (or BLE) security architecture was not designed to protect BLE (or BT, respectively). CTKD enables this capability as the attacker can attack BT from BLE and BLE from BT.

Pairing Intents. The Bluetooth standard should guarantee that a user controls when a device is pairable (or not) and pairs only with devices that she knows about. CTKD violates this expectation as a device is required to be pairable on BT and BLE at the same time and continues to be pairable at least of one of them to support CTKD pairing. CTKD improves the user experience but hides important details about the interplay between BT and BLE and the attacker can take advantage of those.

Key Updates. The Bluetooth standard should guarantee that once two devices are paired their long term keys cannot be changed or at least cannot change without user consent. CTKD not only enables overwriting those keys but also enables reinstalling keys that are weaker than the overwritten ones, which is explicitly forbidden by the standard [12, p. 1401]. For example, the attacker can impersonate a device that supports Numeric Comparison on both transports but while pairing pretending not to support Numeric Comparison.

4 BLUR ATTACKS ON CTKD

We now design and implement four cross-transport attacks based on our security analysis of CTKD (presented in Section 3). Our **BLUR attacks** are the first attacks blurring the security boundary between BT and BLE. The BLUR attacks are standard-compliant and enable impersonation, interception, and manipulation of traffic between victims, as well as unintended sessions with a victim device.

4.1 System Model

Our system model considers two victims, Alice and Bob, who want to securely communicate over BT and BLE. Alice and Bob support Cross-Transport Key Derivation (CTKD) and while pairing and

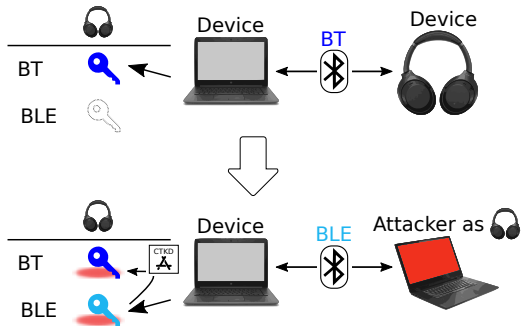

Figure 2: BLUR impersonation attacks. Charlie impersonates a device and triggers the CTKD functionality in order to overwrite an existing pairing key. This enables Charlie to impersonate master and slave devices and take over existing secure BT sessions.

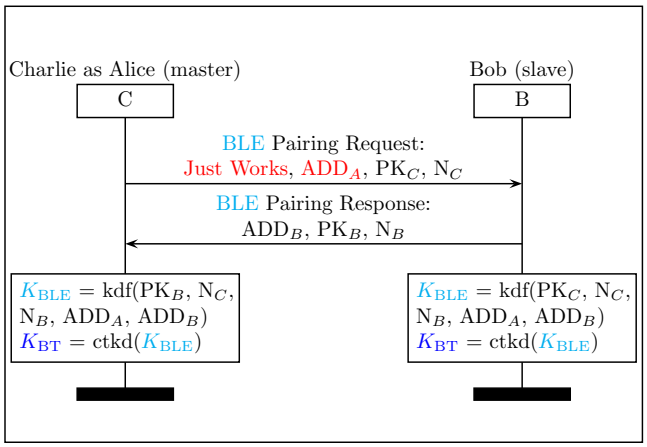

Figure 3: Master impersonation attack and takeover. Charlie (master) sends a BLE pairing request to Bob (slave) as Alice and proposes to use "Just Works". The pairing request includes Alice's Bluetooth address and Charlie's public key and nonce. Bob sends a BLE paring response including his address, public key, and a nonce. Both compute K_{BLE} using the BLE pairing key derivation function (kdf) and derive K_{BT} from K_{BLE} using CTKD's key derivation function (ctkd). As a result, Charlie forces Bob to overwrite the BT and BLE pairing keys that he established with Alice with his keys.

establishing secure sessions, they propose the strongest security mechanisms that they support (e.g., SSP, Secure Connections, or Numeric Comparison association). This setup is supposed to protect Alice and Bob against impersonation, eavesdropping, and man-in-the-middle attacks on BT and BLE. Without loss of generality, we assume that Alice is the master for BT and BLE and Bob is the slave.

Regarding the notation, we indicate a BT long term key with K_{BT}, a BT session key with SK_{BT}, a BLE long term key with K_{BLE}, and a BLE session key with SK_{BLE}. We indicate a Bluetooth address with ADD. Furthermore, we indicate a public key with PK, a private key with SK, a nonce with N, and a message authentication code with MAC.

4.2 Attacker Model and Goals

Our attacker model includes Charlie, a remote attacker in Bluetooth range with the victims. 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 any long term key and session key shared between Alice and Bob and does not observe Alice and Bob when they pair or establish a secure session. Regarding the attacker's capabilities, the attacker can scan and discover BT and BLE devices, jam the Bluetooth spectrum, pair over BT and BLE using CTKD, 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 session.* The second goal is to *impersonate Bob (to Alice) and take over Bob's secure session.* Master and slave impersonations are two different goals as they require different attack strategies. The third goal is to *establish a man-in-the-middle position in a secure session between Alice and Bob.* The third goal requires combining and synchronizing the impersonation attacks on Alice and Bob. The fourth goal is *to pair and establish unintended sessions with Alice or Bob as an arbitrary device, without breaking their bond and secure sessions.*

4.3 Impersonation Attacks and Takeover

The BLUR impersonation attacks include master and slave impersonations and in the remaining of this section, we describe both in detail. For both attacks, we assume that Alice and Bob securely paired over BT in absence of Charlie. Those attacks take advantage of seven cross-transport vulnerabilities that we introduce in Section 3.1 related to pairing, key overwrite, authentication, association, and roles. Section 5.2 presents the implementation of the impersonation attacks.

Figure 2 presents a high-level description of our BLUR impersonation attacks. A laptop and a pair of headphones are paired over BT and run a secure session over BT. Charlie impersonates the headphones and triggers pairing over BLE and CTKD with the laptop. The laptop and the attacker negotiate K_{BLE} (the BLE pairing key) and use CTKD to derive K_{BT} (the BT pairing key). As a result, the laptop overwrites the BT key previously established with the headphones with the BT key established with Charlie. Charlie now effectively takes over the headphone's pairing, and the headphones cannot connect back to the laptop. This attack leverages the fact that the *Security Boundary*, *Pairing Intent*, and *Key Update* security expectations can be violated by CTKD (see Section 3.2).

Master impersonation. Charlie impersonates Alice (master) and takes over her BT secure session with Bob as in Figure 3. Charlie sends a BLE pairing request using Alice's Bluetooth address (ADD_A) and requests to use "Just Works" to avoid user interaction. The BLE pairing request is legal because Charlie impersonates a BLE master. Bob sends a BLE pairing response believing that he is talking to Alice. Charlie and Bob use the exchanged nonces and public keys to

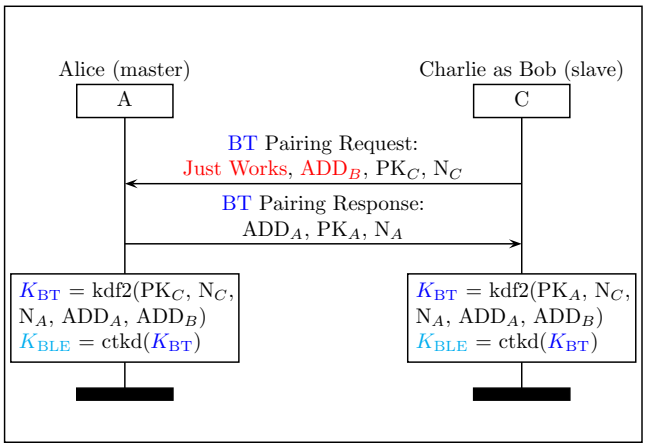

Figure 4: Slave impersonation attack and takeover. Charlie (slave) sends a BT pairing request to Alice (master) as Bob and proposes to use "Just Works". The pairing request includes Bob's Bluetooth address and Charlie's public key and nonce. Alice sends a BT paring response including her address, public key, and a nonce. Both compute K_{BT} using the BT pairing key derivation function (kdf2) and derive K_{BLE} from K_{BT} using CTKD's key derivation function (ctkd). As a result, Charlie forces Alice to overwrite the BT and BLE pairing keys that she established with Bob with his keys.

compute K_{BLE} (kdf) and derive K_{BT} from K_{BLE} using CTKD's key derivation functions (ctkd). As a result of the master impersonation attack, Charlie forces Bob to overwrite the BT and BLE pairing keys that he established with Alice with his keys and takes over Alice. Alice cannot re-establish secure sessions with Bob as she no longer possesses the correct paring keys.

Slave impersonation. Charlie impersonates Bob (slave) and takes over his BT secure session with Alice as in Figure 4. Charlie sends a BT pairing request using Bob's Bluetooth address (ADD$_B$) and requests to use "Just Works" to avoid user interaction. The BT pairing request is legal because BT does not mandate that pairing requests are only sent by a BT master. Alice sends a BT pairing response believing that she is talking to Bob. Charlie and Alice use the exchanged nonces and public keys to compute K_{BT} (kdf2), and derive K_{BLE} from K_{BT} using CTKD's key derivation functions (ctkd). As a result of the slave impersonation attack, Charlie forces

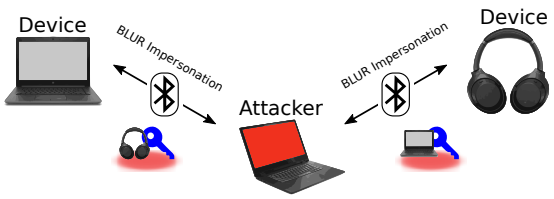

Figure 5: BLUR man-in-the-middle attack. Charlie uses the BLUR Impersonation attack against two devices that where previously paired. The two devices do not detect a change but Charlie now has access to all traffic.

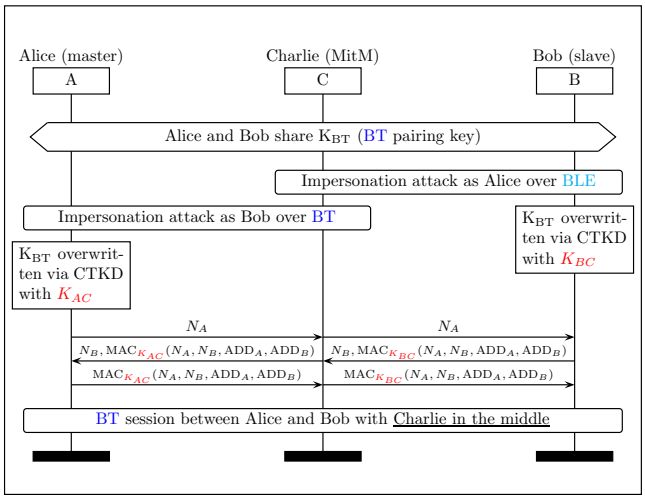

Figure 6: MitM attack and takeover. Charlie impersonates Alice as in Figure 3, impersonates Bob as in Figure 4, let the victims mutually authenticate and then gets access to their traffic.

Alice to overwrite the BT and BLE pairing keys that she established with Bob with his keys and takes over Bob. Bob cannot re-establish secure sessions with Alice as he no longer possess the correct paring keys.

4.4 Man-in-the-Middle Attack and Takeover

Figure 5 presents the high-level description of our BLUR man-in-the-middle attack. As in the previous section, a laptop and a pair of headphones are paired over BT and they run a secure session over BT. Charlie performs the master and slave impersonation attacks described in Section 4.3, overwrites the BT pairing keys of the laptop and the headphones with his keys, and positions himself in the middle between the laptop and the headphones. As a result, Charlie gets access to all traffic between the victims and can inject valid traffic. This attack leverages that the *Security Boundary*, *Pairing Intent* and *Key Update* security expectations can be violated by CTKD (see Section 3.2).

Figure 6 shows the details of the MitM attack. Alice and Bob are paired and share K_{BT}. Firstly, Charlie impersonates Alice to Bob over BLE as in Figure 3 and overwrites Bob's BT key with a new key (K_{BC}) and Alice is disconnected from Bob. Secondly, Charlie impersonates Bob to Alice over BT as in Figure 4 and overwrites Alice's BT key with a new 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 (including the Bluetooth addresses of Alice and Bob). 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.

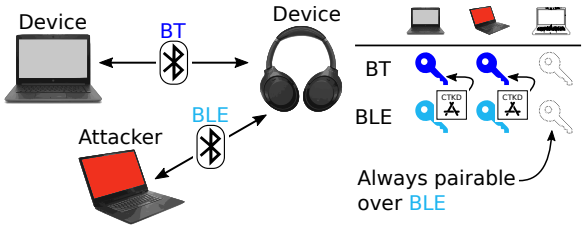

Figure 7: BLUR unintended session attack. In some CTKD use cases a transport is only expected to be used for pairing and is always enabled. For example, a pair of headphones might be always pairable over BLE and expect to only use BT to stream audio. This allows Charlie to initiate pairing as an arbitrary device and establish both BT and BLE keys at will.

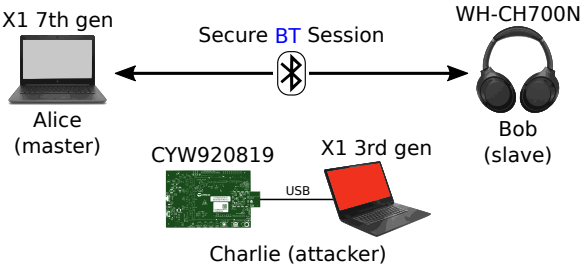

Figure 8: BLUR Attack Scenario. Alice (master) is a ThinkPad X1 7th gen, Bob (slave) is a pair of Sony WH-CH700N headphones and Charlie (attacker) is a CYW920819 board connected via USB to a ThinkPad X1 3rd gen. Alice and Bob have paired in absence of Charlie, and are running a secure BT session.

4.5 Unintended Session Attack

Figure 7 presents a high-level description of the unintended session attack. A laptop and a pair of headphones are paired over BT and run a secure session over BT. Charlie triggers pairing over BLE and CTKD with the headphones as an arbitrary device. As a result, Charlie can establish a secure session over BLE with the headphones without breaking the existing session between the laptop and the headphones, and a secure session over BT when the laptop and the headphones disconnect. Charlie can take advantage of the unintended BT and BLE sessions in several ways. For example, to drop exploits over-the-air such as BlueBorne [6], BLEEDINGBIT [7], or SweynTooth [19], or enumerating and tampering with BT and BLE services and characteristics (including the protected ones). This attack leverages that the *Security Boundary* and *Pairing Intent* security expectations can be violated by CTKD (see Section 3.2).

We note that the unintended session attack is different from an impersonation attack as Charlie does not present himself as a device that was previously trusted by the headphone but as a new device to be trusted. The headphone has no way to recognize Charlie as an attacker can use different identifiers over time such as random Bluetooth names and addresses. However, the unintended session attack technique is similar to the master impersonation attack described in Section 4.3.

5 EVALUATION

To highlight and evaluate the potential of the BLUR attacks, we implement them using open-source software and off-the-shelf hardware. In particular, we describe our attack scenario, our attack device, and how we implement the master impersonation, slave impersonation, man-in-the-middle, and unintentional session BLUR attacks. The design of the attacks is described in detail in Section 4. We also present our implementation of the CTKD key derivation function as specified in the Bluetooth standard that we used to validate the keys generated during our attacks. We will open-source our implementation. We test the impact of the BLUR attack on 13 devices that are all vulnerable to the BLUR attacks.

5.1 Attack Scenario and Attack Device

Figure 8 describes our attack scenario. Alice (master) is represented by a ThinkPad X1 (7th gen) laptop, Bob (slave) is represented by a pair of Sony WH-CH700N headphones and Charlie (attacker) is represented by a CYW920819 Bluetooth development board [16] connected via USB to a ThinkPad X1 (3rd gen) laptop. Alice and Bob have securely paired in absence of Charlie, and are running a secure BT session. The evaluation results are obtained by using the same attack scenario targeting several master and slave devices.

Table 1 presents the relevant Bluetooth features supported by Alice, Bob, and Charlie. We note that Bob is capable of using CTKD over BLE even if he does not support Secure Connections over BT. This confirms the "cross-transport Secure Connections" vulnerability that we discuss in Section 3.1. In Table 1, we append an asterisk (*) to the attacker's features that we can modify with our implementation.

Our attack device consists of a Linux laptop (Bluetooth host) connected 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 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 BT and BLE.

Regarding the controller, we use the 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 1. This process required significant engineering effort as we had to dump the Bluetooth firmware from the board, reverse-engineer the relevant functions and data structures, and write and test our ARM assembly patches.

Our attack device makes use of several free and open-source tools to automate the configuration and management of BT, BLE, and the BLUR attacks. Table 2 presents the list of such tools with a brief description of their usage. Overall, our usage of low-cost hardware and open-source software should enable other researchers to easily reproduce the BLUR attacks.

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

Table 1: Relevant Bluetooth features for Alice, Bob, and Charlie. 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.

Software/Tool	Usage
`ghidra`	RE the devboard firmware [35]
`internalblue`	Patch devboard firmware [27]
`wireshark`	Monitor HCI, LMP, and SMP
`hciconfig`	Configure HCI interfaces
`hcitool`	Scan, connect and enumerate BLE devices
`bleah`	Scan, connect and enumerate BLE devices
`scapy`	Craft and decode packets [11]
`pybt`	Custom BLE pairing [32]
`linux414`	Modify BLE pairing capabilities
`bluez`	Modify Linux userspace configuration
`pybluez`	Test BT and BLE using the BlueZ API
`scapy`	Configure HCI, manage BT and BLE sockets
`bluetoothctl`	Manage, pair and connect devices
`btmgmt`	Manage, pair and connect devices

Table 2: Open-source software and tools used to implement the BLUR attacks.

5.2 Implementation of the BLUR 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 using our attack device in the attack scenario presented in Section 5.1.

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 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.

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.

Unintended session attack. To perform the unintended session 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.

5.3 CTKD KDF Implementation

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]

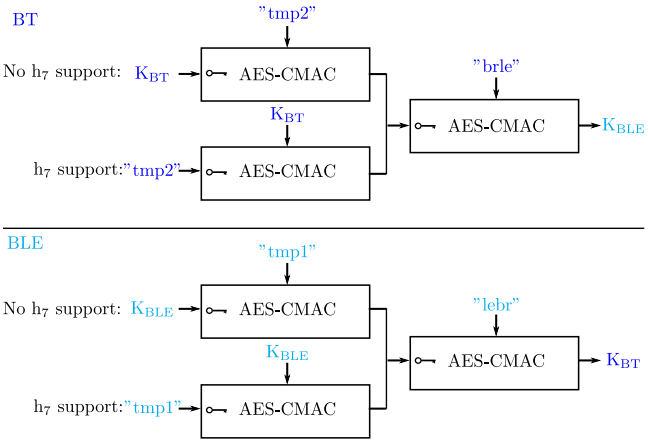

Figure 9: CTKD key derivation function for BT (top) and BLE (bottom). BT and BLE use a chain of two AES-CMAC with different keys and 4-byte constant strings. BT uses K_{BT}, "tmp2" and "brle" and derives K_{BLE}. BLE uses K_{BLE}, "tmp1" and "lebr" and derives K_{BT}. In the first AES-CMAC, If both devices 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.

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.

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 9 shows the CTKD key derivation function for BT (top) and BLE (bottom). Both use 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 "lebr" and derives K_{BT}.

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.

5.4 Evaluation Setup

With our attack implementation (Section 5.2), we are capable of conducting the 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 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 such 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.

5.5 Evaluation Results

We evaluated the BLUR attacks on 13 devices, and Table 3 shows our evaluation results. The first six columns indicate the device producer, device model, OS, chip manufacturer, chip model, and supported Bluetooth version. The seventh column indicates the attacker role. The last three columns contain a checkmark (✓) if a device is vulnerable to the master Impersonation attack (MI), slave impersonation attack (SI), man-in-the-middle attack (MitM), or unintended session (US) attack. 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 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 evaluation demonstrates that the BLUR attacks are practical, standard-compliant, and affects all the Bluetooth versions that support CTKD (i.e., Bluetooth versions ≥ 4.1). As the BLUR attacks are standard-compliant, potentially all standard-compliant devices supporting CTKD are also vulnerable. Based on our evaluation, we suggest the Bluetooth SIG to fix the vulnerabilities that we uncover in CTKD and we provide our set of countermeasures for the Bluetooth standard in Section 6.2.

6 DISCUSSION

We now discuss the main lessons learned while analyzing CTKD, our set of countermeasures to mitigate the BLUR attacks, and how to guarantee the security expectations mentioned in Section 3.2.

6.1 Lesson Learned from CTKD

One key lesson that we learned while analyzing CTKD is that combining protocols with different security architectures and threat models, such as BT and BLE, might introduce standard-compliant cross-transport vulnerabilities. Such vulnerabilities are very effective and difficult to isolate as they manifest at the security boundary between the combined protocols.

In particular, separate security analyses of the combined technologies are insufficient to discover cross-transport vulnerabilities. Such vulnerabilities require a security analysis that considers both

Device			Chip		Bluetooth	BLUR Attack			
Producer	Model	OS	Producer	Model	Version	Role	MI/SI	MitM	US
Cypress	CYW920819EVB-02	Proprietary	Cypress	CYW20819	5.0	Slave	✓	✓	✓
Dell	Latitude 7390	Win 10 PRO	Intel	8265	4.2	Slave	✓	✓	✓
Google	Pixel 2	Android	Qualcomm	SDM835	5.0	Slave	✓	✓	✓
Lenovo	X1 (3rd gen)	Linux	Intel	7265	4.2	Slave	✓	✓	✓
Lenovo	X1 (7th gen)	Linux	Intel	9560	5.1	Slave	✓	✓	✓
Samsung	Galaxy A40	Android	Samsung	Exynos 7904	5.0	Slave	✓	✓	✓
Samsung	Galaxy A51	Android	Samsung	Exynos 9611	5.0	Slave	✓	✓	✓
Samsung	Galaxy A90	Android	Qualcomm	SDM855	5.0	Slave	✓	✓	✓
Samsung	Galaxy S10	Android	Broadcom	Exynos 9820	5.0	Slave	✓	✓	✓
Samsung	Galaxy S10e	Android	Broadcom	Exynos 9820	5.0	Slave	✓	✓	✓
Samsung	Galaxy S20	Android	Broadcom	Exynos 990	5.0	Slave	✓	✓	✓
Sony	WH-1000XM3	Proprietary	CSR	12414	4.2	Master	✓	✓	✓
Sony	WH-CH700N	Proprietary	CSR	12942	4.1	Master	✓	✓	✓

Table 3: BLUR attacks evaluation results. The first six columns indicate the device producer, device model, OS, chip manufacturer, chip model, and supported Bluetooth version. The seventh column indicates the attacker role. The last three columns contain a checkmark (✓) if a device is vulnerable to the master Impersonation attack (MI), slave impersonation attack (SI), man-in-the-middle attack (MitM), or unintended session (US) attack. 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 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 relevant BLUR attacks.

technologies and related threat models at the same time. Such an analysis must take into account an attacker that is able to use and mix features from both technologies, and exploit one technology taking advantage of vulnerabilities in the other one.

6.2 Countermeasures

Mitigating the BLUR attacks requires changes at the Bluetooth standard level. The different countermeasures have different trade-offs regarding complexity and provided guarantees. All proposed countermeasures can be implemented in the Bluetooth host (OS) by storing extra metadata about a trusted Bluetooth device and by using available HCI commands and events.

Disable CTKD key overwrite. CTKD allows to write and overwrite BT and BLE long term keys (which violate the Security Boundary, Pairing Intents, and Key Updates security expectation introduced in Section 3.2). This issues can be exploited to take over legitimate sessions from a victim device by impersonating the victim and using CTKD to overwrite their long term key with the attacker long term key. To fix this issue, a device should disallow key overwrites with CTKD when a paired device wants to re-pair. This can be accomplished during re-pairing by not running CTKD and continue using the trusted long term key that was already established in a previous pairing session.

Enforce strong association mechanisms. CTKD allows two devices to use different association mechanisms on different transports when pairing and re-pairing (which violates the Key Update security expectation). The BLUR attack exploits this fact to re-pair with a victim device using "Just Works" even if the victim does support Numeric Comparison. To fix this issue, a device should keep track of which BT and BLE keys are established using CTKD, record the strongest association mechanism used while pairing and enforce it for subsequent pairings.

Enforce Secure Connections. 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 supported on both BT and BLE. To fix this issue, a device should enforce that "Secure Connections" is supported on BT and BLE before running CTKD. If "Secure Connections" is not supported then the device must raise an error instead.

CTKD Notifications. CTKD is completely transparent to the end-user and is specified in the standard as an optional feature. We exploit those facts to improve the stealthiness of our attacks, violating the Pairing Intent security expectation. Given that CTKD is a security-critical feature we believe that CTKD should not be considered optional, and a device should notify the user every time such a feature is in use. For example, the device should notify the user when he is re-pairing with a trusted device and is using CTKD to overwrite a long term key.

7 RELATED WORK

Bluetooth standard compliant attacks are particularly dangerous as all Bluetooth devices are affected, regardless of version numbers or implementation details. Such standard-compliant attacks have appeared since the first versions of Bluetooth [23, 26]. Standard-compliant attacks on BT include attacks on legacy pairing [33], SP [10, 20, 34] association [21], key negotiation [2], and authentication procedures [3, 25, 36]. Standard-compliant attacks on BLE include attacks on legacy pairing [31], key negotiation [4], SSP [10], and GATT [24] Compared to the mentioned attacks that target either BT or BLE, the BLUR attacks are the first attacks targeting the intersection between BT and BLE. In addition, the BLUR attacks are also standard-compliant.

We have seen attacks targeting specific implementation flaws on BT [6] and BLE [7, 19]. 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, 28, 29] but none of those surveys (and the Bluetooth standard) is considering 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 [1]. Our BLUR attacks are the first cross-transport attacks for BT and BLE.

8 CONCLUSION

In this work, we present novel and standard-compliant vulnerabilities affecting Bluetooth CTKD and attacks taking advantage of such vulnerabilities. CTKD is a feature enabling two devices to securely use BT and BLE by just pairing over either and then deriving the keying material for the other. CTKD is security-critical as it enables to establish (and overwrite) BT and BLE long term keys, while using the strongest Bluetooth security mode (e.g., Secure Connections). Despite that, the Bluetooth standard does not provide a security evaluation of CTKD and does not include it in the BT and BLE threat models.

To address those issues we performed the first security analysis of CTKD. Our analysis uncovered seven cross-transport vulnerabilities affecting mechanisms such as pairing, key derivation, authentication, and association. Furthermore, we show how to exploit such vulnerabilities to perform novel cross-transport attacks. We name our attacks BLUR attacks, as they blur the security boundary between BT and BLE. Our attacks enable exploiting BLE from BT and BT from BLE in several ways. For example, an attacker can perform cross-transport master and slave impersonation attacks to take over existing sessions between the impersonated victim and another victim, and combine them to mount a cross-transport man-in-the-middle attack.

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 13 devices from different hardware and software manufacturers. Our devices range across all the Bluetooth versions supporting CTKD (version greater or equal to 4.1). As the BLUR attacks are standard-compliant, all devices supporting CTKD become potentially vulnerable. We sketch a set of countermeasures to address the BLUR attack directly in the Bluetooth standard. The countermeasures generally require to keep additional state about paired devices. We have disclosed our findings and our countermeasures to the Bluetooth SIG in May 2020.

REFERENCES

[1] 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)*.

[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. 2020. BIAS: Bluetooth Impersonation AttackS. In *2020 IEEE Symposium on Security and Privacy*. IEEE.

[4] 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

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

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

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

[8] Python Cryptographic Authority. 2019. 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. 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.

[10] 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, Accessed: 2018-10-30. (2018).

[11] Philippe Biondi. 2018. Scapy: Packet crafting for Python2 and Python3. https://scapy.net/, Accessed: 2018-01-26. (2018).

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

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

[14] BlueZ. 2014. Bluetooth 4.2 features going to the 3.19 kernel release. http://www.bluez.org/bluetooth-4-2-features-going-to-the-3-19-kernel-release/, Accessed: 2020-01-27. (2014).

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

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

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

[18] Morris Dworkin. 2018. 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] Garbelini, Matheus and Chattopadhyay, Sudipta and Wang, Chundong. 2020. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://asset-group.github.io/disclosures/sweyntooth/sweyntooth.pdf, Accessed: 2020-04-08. (2020).

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

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

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

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

[24] Sławomir Jasek. 2016. Gattacking Bluetooth smart devices. Black Hat USA Conference. (2016).

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

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

[27] 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.

[28] 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.

[29] John Padgette. 2017. Guide to bluetooth security. *NIST Special Publication* 800 (2017), 121.

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

[31] Mike Ryan. 2013. Bluetooth: With Low Energy Comes Low Security. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, Vol. 13. USENIX, 4–4.

[32] Mike Ryan. 2015. PyBT: Hackable Bluetooth stack in Python. https://github.com/mikeryan/PyBT, Accessed: 2019-06-19. (2015).

[33] 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.

[34] 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.

[35] National Security Agency USA. 2019. 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).

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

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