BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses

Daniele Antonioli
EURECOM
Sophia Antipolis, France
daniele.antonioli@eurecom.fr

ABSTRACT

Bluetooth is a pervasive technology for wireless communication. Billions of devices use it in sensitive applications and to exchange private data. The security of Bluetooth depends on the Bluetooth standard and its two security mechanisms: pairing and session establishment. No prior work, including the standard itself, analyzed the *future and forward secrecy* guarantees of these mechanisms, e.g., if Bluetooth pairing and session establishment defend past and future sessions when the adversary compromises the current. To address this gap, we present *six* novel attacks, defined as the *BLUFFS attacks*, breaking Bluetooth sessions' forward and future secrecy. Our attacks enable device impersonation and machine-in-the-middle *across* sessions by only compromising *one* session key. The attacks exploit *two* novel vulnerabilities that we uncover in the Bluetooth standard related to unilateral and repeatable session key derivation. As the attacks affect Bluetooth at the architectural level, they are effective regardless of the victim's hardware and software details (e.g., chip, stack, version, and security mode).

We also release `BLUFFS`, a low-cost toolkit to perform and automatically check the effectiveness of our attacks. The toolkit employs *seven* original patches to manipulate and monitor Bluetooth session key derivation by dynamically patching a closed-source Bluetooth firmware that we reverse-engineered. We show that our attacks have a *critical* and *large-scale* impact on the Bluetooth ecosystem, by evaluating them on *seventeen* diverse Bluetooth chips (*eighteen* devices) from popular hardware and software vendors and supporting the most popular Bluetooth versions. Motivated by our empirical findings, we develop and successfully test an *enhanced* key derivation function for Bluetooth that stops *by-design* our six attacks and their four root causes. We show how to effectively integrate our fix into the Bluetooth standard and discuss alternative implementation-level mitigations. We *responsibly disclosed* our contributions to the Bluetooth SIG.

CCS CONCEPTS

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

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623066

KEYWORDS

Bluetooth, forward secrecy, future secrecy, attacks, defenses

ACM Reference Format:
Daniele Antonioli. 2023. BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623066

1 INTRODUCTION

Bluetooth is a *pervasive* technology for low-power wireless communication [10, 12, 13]. It provides two transports: Bluetooth Classic for high throughput and connection-oriented use cases and Bluetooth Low Energy (BLE) for connectionless and low throughput scenarios. This paper focuses on *Bluetooth Classic*, from now indicated as *Bluetooth*. As billions of devices, such as smartphones, laptops, speakers, headsets, and tablets, daily employ Bluetooth to exchange sensitive data and commands, Bluetooth must provide strong security and privacy guarantees, including confidentiality, integrity and authenticity.

Bluetooth's security and privacy depend on *pairing* and *session establishment*, two mechanisms specified in the *Bluetooth standard* (v5.3) [11]. Devices use pairing to agree upon a long-term secret called the pairing key. Pairing involves user interaction, such as pressing a button or confirming a numeric value on the screen. Paired devices use session establishment to create encrypted and integrity-protected connections, each protected by a *fresh* session key derived from the (static) pairing key and runtime parameters (key diversifiers). Session establishment, unlike pairing, does *not* require user interaction. These two mechanisms have two security modes: (i) *Legacy Secure Connections (LSC)* using legacy cryptographic primitives and procedures, (ii) *Secure Connections (SC)* employing FIPS-compliant ones, such as ECDH, AES-CCM. Pairing and session establishment are *critical* attack surfaces as if they are vulnerable, an adversary can exploit such vulnerability on any (standard-compliant) Bluetooth device. This critical risk motivated extensive research on pairing [4, 9, 25, 29, 37, 53, 56] and session establishment [3, 5] (see Section 8 for more works).

But, no prior work has investigated Bluetooth's *forward* and *future secrecy* guarantees and their relation with pairing and session establishment. Forward and future secrecy, which enable to defend past and future messages from key compromise attacks, are *not* even discussed by the Bluetooth standard. We extrapolated these properties via a careful analysis of the standard. We inferred that Bluetooth should provide forward and future secrecy among sessions if the pairing key stays secret. Hence, an attacker compromising the current session key should not be able to decrypt

data from past (i.e., forward secrecy) and future sessions (i.e., future secrecy). Then we questioned this assumption and uncovered that, instead, sessions' forward and future secrecy *can* be broken by stealthily attacking *session key derivation* at the protocol level, *without* knowing the pairing key or triggering a new (suspicious) pairing event.

Specifically, we present the **BLUFFS attacks**, six novel attacks breaking Bluetooth's forward and future secrecy by targeting session establishment. The attacks exploit an attack strategy forcing LSC session establishment and manipulating in novel ways its key derivation to *reuse* a key known to the attacker across sessions. The attacker first installs a weak session key, then spends some time brute-forcing it, and reuses it to impersonate or machine-in-the-middle (MitM) a victim in subsequent sessions (breaking future secrecy) and decrypt data from past sessions (breaking forward secrecy). We decline the attack strategy in six attack scenarios related to the victim's connection role (i.e., initiator or responder) and Bluetooth security mode (i.e., LSC or SC). Moreover, we detail the *four* attacks' root causes, two of which uncover that the standard allows to *unilaterally* derive session keys *without* relying on nonces.

We develop the `BLUFFS` toolkit to perform and detect the BLUFFS attacks automatically and with low effort. The toolkit provides an *attack device* module requiring open-source software, a Linux laptop, and a Cypress/Infineon CYW20819 board [30]. We provide *seven* new patches for the board's closed-source firmware enabling monitoring and tampering with Bluetooth session key derivation. Moreover, our *attack checker* module cleverly parses and analyzes session establishment messages, aka Link Manager Protocol (LMP) packets from a pcap file to automatically compute session keys and detect our attacks.

We demonstrate that the BLUFFS attacks are *effective on a large scale* by evaluating eighteen devices embedding seventeen unique Bluetooth chips. We successfully exploited a broad set of devices (e.g., laptops, smartphones, headsets, and speakers), operating systems (e.g., iOS, Android, Linux, Windows, and proprietary OSes), Bluetooth stacks (e.g., BlueZ, Gabeldorsche, Bluedroid, and proprietary ones), vendors (e.g., Intel, Broadcom, Cypress, Cambridge Silicon Radio, Infineon, Bestechnic, Apple, Murata, Universal Scientific Industrial, Samsung, Dell, Google, Bose, Logitech, Xiaomi, Lenovo, Jaybird, and Qualcomm), and Bluetooth versions (e.g., 5.2, 5.1, 5.0, 4.2, and 4.1).

Motivated by our evaluation results, we propose an *enhanced* Bluetooth session key derivation function that *stops by-design* our attacks and their root causes. Our countermeasure is *backward compatible* with the Bluetooth standard and adds minimal overheads. Specifically, it reuses standard-compliant crypto primitives (i.e., e_1 and e_3) and link-layer functions (i.e., LMP commands). It requires forty-eight (48) extra bytes over the air and three extra function calls. We successfully test the fix against the BLUFFS attacks at the protocol level and release the fix in our toolkit. We also discuss implementation-specific mitigations that vendors can use to mitigate some BLUFFS attacks.

We summarize our contributions as follows:

- We study Bluetooth's forward and future secrecy guarantees, two essential properties currently not discussed by prior work and the Bluetooth standard. We show six novel attacks, named BLUFFS attacks, breaking these properties by exploiting Bluetooth's session key derivation. The threats enable device impersonation and MitM across sessions by only compromising one session key. They do not require user interaction or compromise Bluetooth pairing (keys). The attacks are *specification-compliant* as they target protocol-level weaknesses in the Bluetooth standard. We discuss the four attacks' root causes, two of which are novel for Bluetooth and affect session key derivation. We also explain how the attacks extend the state-of-the-art, including the KNOB and BIAS attacks [3, 5].
- We release `BLUFFS`, a low-cost and reproducible toolkit to perform and automatically check our attacks. The toolkit's *attack device* enables manipulation and monitoring of Bluetooth session key derivation. The toolkit's *attack checker* uses a novel LMP parsing and analysis strategy to detect our attacks from a pcap file automatically. Our toolkit complements and extends the state of the art of Bluetooth security testing, such as [17, 18, 47].
- We tested the six BLUFFS attacks on *eighteen* devices embedding *seventeen* different Bluetooth chips from popular hardware and software vendors. The attacks are *successful* against all six LSC chips with one exception and against all eleven SC chips when the impersonated victim is an LSC device. If both victims support SC, the attacks are effective on two out of eleven victims. From our empirical result we conclude that the BLUFFS attacks are *practical* and a *critical risk* for the Bluetooth ecosystem, and should be fixed with high priority.
- We design a backward-compliant Bluetooth session key derivation function based on *fresh*, *authenticated*, and *mutual* key derivation. Our function stops the six BLUFFS attacks and addresses their four root causes at the protocol level. We show how to integrate our countermeasure into the Bluetooth standard with minimal overhead (e.g., three LMP packets and three function calls). We also present our successful evaluation of the fix against our attacks at the protocol level and release it as part of our `BLUFFS` toolkit.

Responsible disclosure. We responsibly disclosed our findings and toolkit to the Bluetooth Special Interest Group (SIG) [14] in October 2022. The Bluetooth SIG acknowledged our findings, coordinated the disclosure with the affected vendors, and reserved *CVE-2023-24023* for our report. We also reached out to Google, Intel, Apple, Qualcomm, and Logitech. Google scored our report with high severity, awarded us a bounty, and is working on a fix. Intel did the same but scored the report with medium severity. Apple and Logitech acknowledged the report and are working on fixes. Qualcomm has not replied yet. We *anonymously* release in a *private* repository our toolkit's attack checker, countermeasure, and part of the attack device at https://anonymous.4open.science/r/sec23-anon-654A. We will *open-source* the toolkit according to responsible disclosure, and will submit it for *artifact evaluation*.

2 PRELIMINARIES

We present the required Bluetooth preliminaries and our extrapolation of Bluetooth's forward and future secrecy guarantees from the Bluetooth standard.

2.1 Bluetooth

Bluetooth is the *de-facto standard* technology for low-power and reliable wireless communication and has an open specification (v 5.3) [11]. It was born as a cable-replacement wireless protocol for the unlicensed 2.4 GHz ISM (Industrial, Scientific and Medical) band, and evolved to address various use cases requiring high-throughput and persistent connections. For example, it supports wireless audio streaming, file transfer, hands-free services, peer-to-peer connections, and Internet bridging. Bluetooth packets should be protected against relevant attacks, such as device spoofing and MitM, as it transports sensitive data and commands.

The Bluetooth stack loosely follows the Open Systems Interconnection (OSI) model. At the physical layer, it employs synchronized frequency hopping and time division multiple access. The link layer uses a star topology managed by the link manager protocol (LMP). The link layer connection initiator is known as *Central*, while the responder is called *Peripheral*. These two roles can be switched dynamically during connection establishment or while a connection is ongoing. Bluetooth uses a six-byte, unique, and static address to identify a device at the link layer. A Bluetooth address does not contain secret information and is obtained with an inquiry procedure. At the application layer, Bluetooth provides several *profiles*, such as the advanced audio distribution (A2DP) profile. The Bluetooth *Controller* manages the physical and link layers, while the Bluetooth *Host* takes care of the upper layers. The Host and the Controller communicate via the *Host Controller Interface (HCI)*, a protocol based on commands and events.

The Bluetooth standard specifies *link-layer security mechanisms*, providing confidentiality, integrity, and authenticity to upper layers, including all Bluetooth profiles. *Pairing* allows devices to establish a long-term pairing key (PK). The standard defines such a procedure as Secure Simple Pairing (SSP) [11, p. 268]. *Session establishment* enables paired devices to establish a secure session using a fresh session key (SK). SK is derived from PK and constant and variable inputs. The standard includes two security modes affecting pairing and session establishment: LSC, which employs legacy security mechanisms for backward-compliance reasons (e.g., E_0 and SAFER+), and SC that uses FIPS-compliant ones (e.g., ECDH, AES-CCM, and HMAC).

2.2 Bluetooth Forward and Future Secrecy

Despite their critical associated risks, Bluetooth's forward and future (i.e., backward) secrecy guarantees are *unexplored*. By compromising forward (future) secrecy, the attacker could break the confidentiality of past (future) sessions. However, we do not know if these attacks and vulnerabilities exist as the Bluetooth standard neither covers nor define forward and future secrecy, and no prior research investigated them. In this work, we address this crucial gap.

We examined pairing and session establishment from the standard and extracted their forward and future secrecy guarantees.

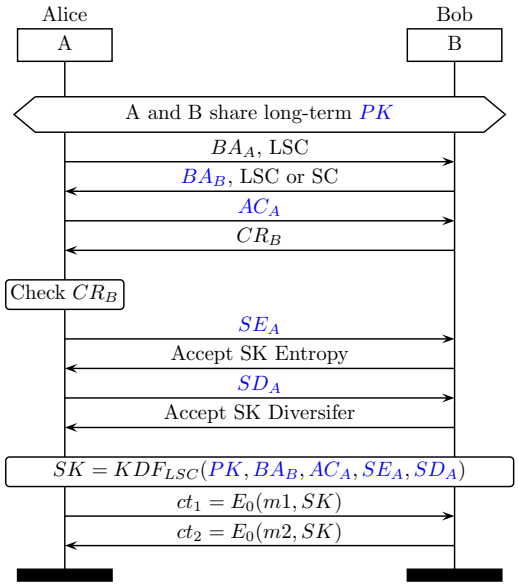

Figure 1: Bluetooth LSC session establishment. The values in blue are used to compute a fresh SK. PK and BA_B are *constant*, while AC_A, SE_A, and SD_A are *variable*. In this example, Alice (i.e., the Central) controls SK derivation as she provides all the variable SK derivation inputs.

Bluetooth *should* provide forward and future secrecy *across sessions* until PK or the SK key derivation function (KDF) are not compromised. Specifically, an attacker compromising the current SK cannot target past and future sessions because each session employs a fresh (i.e., different) SK derived from PK and variable key diversifiers. So it is crucial that PK stays secret and that SK is properly derived. Nevertheless, no prior work evaluated the strength of SK derivation and the existence of related (practical and impactful) attack scenarios.

Now we describe *LSC* session establishment, including its key derivation phase, as is the target of our work. We assume that Alice (Central with address BA_A) and Bob (Peripheral with address BA_B) are paired and share PK. As shown in Figure 1, LSC session establishment starts with two messages where Alice and Bob identify themselves and negotiate LSC. Then, Alice asks Bob to authenticate PK by sending a challenge AC_A. Bob sends back CR_B, a response computed from PK and AC_A, and Alice checks that CR_B equals the response she computed locally. Then, Alice sends SE_A, an SK entropy proposal between 16 and 7 bytes, and Bob can accept it (as in Figure 1) or propose a lower value to be accepted by Alice. Once SK entropy negotiation is completed, Alice sends to Bob SD_A, a *session key diversifier*, and Bob acknowledges it. Finally, the devices use KDF_{LSC} to derive SK from *variable* (AC_A, SE_A, SD_A) and *constant* inputs (PK, BA_B).

KDF_{LSC} is the LSC key derivation function specified in the standard and is defined as a system of three equations [11, p. 267]:

$$COF = e_1(PK, AC_A, BA_B) \quad (1a)$$

$$ISK = e_3(PK, SD_A, COF) \quad (1b)$$

$$SK = e_s(ISK, SE_A) \quad (1c)$$

Using Equation 1a, the devices compute a ciphering offset number (*COF*) from the pairing key, Alice's authentication challenge and Bob's Bluetooth address. The computation uses the e_1 authentication function [11, p. 975], which is based on the SAFER+ block cipher [40]. Then, an intermediate session key (*ISK*) is computed via Equation 1b, using the pairing key, Alice's session key diversifier, and COF. The second computation employs the e_3 key generation function [11, p. 981]. Finally, Alice and Bob derive *SK* by reducing the entropy of *ISK* according to SE_A with the e_s function as shown in Equation 1c. The reduction function relies on on modular arithmetic over polynomials in the finite Galois field [8].

3 THREAT MODEL

Here we present the paper's system and attacker models and our notation.

3.1 System Model

Our system model considers a scenario where Alice and Bob (i.e., the victims) want to communicate securely using Bluetooth. Alice and Bob represent arbitrary devices (e.g., laptops, headsets, and smartphones) and can employ any Bluetooth profile (e.g., audio, hands-free, and Internet bridge). We assume the victims have already *paired* using their strongest security capabilities (e.g., SSP and SC).

The paired victims establish secure connections using Bluetooth's *session establishment*. Alice is the Central (initiator) and Bob the Peripheral (responder), unless stated otherwise. As discussed in Section 2.2, if an attacker compromises the current *SK*, she should be unable to compromise past and future sessions (i.e., break forward and future session secrecy), as each session employs a fresh (i.e., different) *SK*.

3.2 Attacker Model

Our attacker model considers Charlie, a *proximity-based* attacker in Bluetooth range with the victim(s). The attacker can capture Bluetooth packets in plaintext (e.g., authentication challenges, key diversifiers, and negotiated entropy values) and ciphertext (e.g., encrypted audio, files, or internet traffic). Charlie knows the victim's Bluetooth address, can craft (standard-compliant) Bluetooth packets, and negotiate arbitrary capabilities. Charlie cannot compromise *PK*, does not observe the victims while they are pairing, and does not trigger new pairing events. She cannot tamper with the victims' devices, including their hardware and software components. The attacker can downgrade the entropy of *SK* to the lowest value supported by a victim (e.g., 1 byte for devices not patched against the KNOB attack or 7 bytes) and brute force *SK*. We do not assume a specific brute-force effort to cover attackers with different capabilities and resources (e.g., motivated and average attackers).

Charlie wants to break the forward and future secrecy of Alice and Bob' sessions. For example, she would like to impersonate Alice to Bob, Bob to Alice, or MitM them *across* sessions to decrypt past messages (i.e., breaking forward secrecy) and decrypt or inject future ones (i.e., compromising future secrecy). These goals are novel as the state-of-the-art assumes an adversary targeting the *current* session (e.g., KNOB [5] and BIAS [3]). Moreover, the attacker would like to exploit *any* Bluetooth device, regardless of its Bluetooth capabilities (e.g., chip, version, software stack, security mode, and supported profiles).

3.3 Notation

In the paper, we use the following notation. We indicate a Bluetooth address as *BA*, an authentication challenge as *AC* (AU_RAND in the standard), a challenge-response as *CR* (SRES in the standard), a session key as *SK* (Kc' in the standard), a pairing key as *PK* (LK in the standard), a session key entropy proposal as *SE* and a session key diversifier as *SD* (EN_RAND in the standard). We abbreviate a key derivation function with *KDF*. We use A, B, and C subscripts to indicate Alice, Bob, (the victims) and Charlie (the attacker).

4 BLUFFS ATTACKS

In this section, we describe the *BLUFFS attacks*, *six* new threats breaking Bluetooth's forward and future secrecy and enabling impersonation and MitM attacks *across* sessions. We also present the *four* attacks' root causes related to *SK* derivation during session establishment and explain why our attacks *extend* the state of the art (e.g., KNOB [5] and BIAS [3]). Please refer to Section 2 for the attacks' preliminary and Section 3 for their threat model.

4.1 Attack Description

Strategy. The BLUFFS attacks take advantage of a *novel* attack strategy, enabling Charlie to reuse a weak session key (SK_C) across sessions to spoof or MitM arbitrary victims (e.g., LSC and SC Centrals and Peripherals). We now describe such a strategy in an impersonation attack setup with the help of Figure 2. Charlie presents to Bob using Alice's Bluetooth address (BA_A) obtained using Bluetooth inquiry procedures or de-anonymization attacks such as [16]. She negotiates LSC mode (LSC) to force LSC session establishment (and key derivation), whether Bob supports LSC or SC. If Charlie is a Peripheral, she switches to the Central role to lead session establishment, including *SK* derivation. As a consequence, Charlie can target Bob as a Central (initiator) or a Peripheral (responder).

Next, Charlie forces a *fixed* and *weak* session key (SK_C) by cleverly negotiating her session key derivation parameters. Specifically, she sends a constant authentication challenge (AC_C) and ignores Bob's response (CR_C). She proposes the lowest session key entropy value (SE_C) to (re)establish a weak key and a *constant* session key diversifier SD_C. As a result, Bob (re)derives SK_C by using KDF_{LSC} with constant inputs, i.e., *PK*, BA_B, AC_C, SE_C, and SD_C. For example, Charlie can set AC_C and SD_C equal to zero, and SE_C equal to one (SK_C has one byte of entropy).

We employ our attack strategy in *six* attacks covering all combinations of impersonation and MitM attacks across sessions (i.e., targeting SC and LSC Centrals and Peripherals). As shown in the following enumeration, the attacker can spoof a LSC Central or Peripheral to a LSC or SC victim (i.e., A1, A2), impersonate a SC Central or Peripheral to a LSC or SC victim (i.e., A4, A5), or MitM

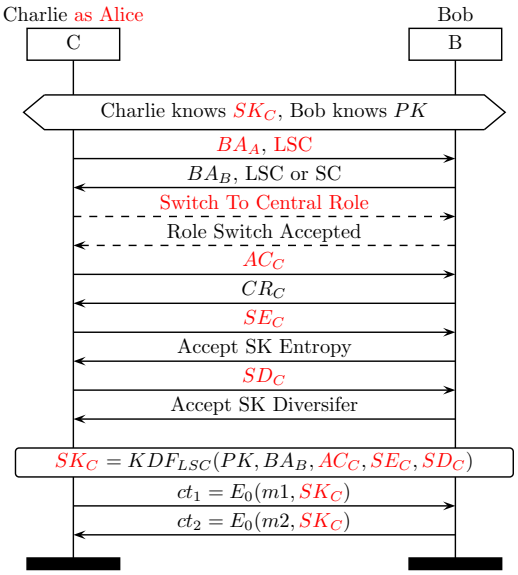

Figure 2: BLUFFS attacks strategy. Charlie approaches Alice as Bob negotiates LSC regardless of the security mode supported by Bob, and, if she is a Peripheral, switches to the Central role. Then, during LSC key derivation, she proposes *constant* values (AC_C, SE_C, SD_C) to force the derivation of a *fixed* session key (SK_C). Charlie employs this strategy while impersonating (or MitMing) Alice and Bob to *reuse* SK_C across sessions.

a session where one victim supports LSC or both victims support SC (i.e., A3, A6).

- A1: Spoofing a LSC Central to a victim Peripheral
- A2: Spoofing a LSC Peripheral to a victim Central
- A3: MitM session where one victim supports LSC
- A4: Spoofing a SC Central to a victim Peripheral
- A5: Spoofing a SC Peripheral to a victim Central
- A6: MitM session where the victims support SC

The BLUFFS attacks break Bluetooth's session forward and future *without* compromising prior (strong) *SK*s negotiated by the victims. We consider forward (future) secrecy broken if Charlie compromises past (future) sessions once SK_C is brute-forced (i.e., compromised). As shown by the timeline in Figure 3, the attacker at time t_1 mounts a MitM attack forcing SK_C (A3 or A6), captures the traffic on the current and subsequent sessions, and starts brute forcing SK_C. At $t_2 > t_1$ she brute forces (compromises) SK_C and decrypts all past messages exchanged since t_1 violating forward secrecy. At $t_3 > t_2$ she reuses SK_C to impersonate or MitM Alice and Bob across the next sessions (A1, A2, A3, A4, A5, and A6). Hence, she breaks future secrecy by violating the sessions' confidentiality, integrity, and authenticity from t_2 onwards.

Brute force setup and effort. Charlie brute forces SK_C employing an *offline* and *parallelizable* setup similar to [5]. She tests offline multiple session keys against one or more sniffed ciphertexts using known Bluetooth packet fields as oracles (e.g., L2CAP and RFCOMM headers decrypting to known constants). The attacker's brute force

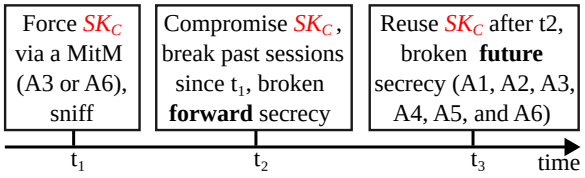

Figure 3: BLUFFS attacks timeline. The attacker forces SK_C at time t_1 via a MitM attack (A3 or A6) and sniffs the messages exchanged by the victims. She compromises (brute forces) SK_C at time t_2 and breaks forward secrecy by decrypting past traffic since t_1. She reuses SK_C at time t_3 to impersonate or MitM a victim (A1, A2, A3, A4, A5, or A6) and compromises future secrecy.

effort is proportional to SE_C (i.e., *SK*'s entropy). However, it does *not* depend on the number of targeted sessions as it should with proper forward and future secrecy mechanisms. If SE_C is one, the brute force effort is negligible, i.e., 128 trials on average within a key space of 256 elements. Otherwise, if it is seven, the attacker requires 2^55 trials on average within a key space of 2^56 elements. Based on prior work breaking symmetric cryptosystems with seven bytes of entropy, such as the data encryption standard (DES) [22, 35], we estimate a moderate effort for a low-cost attacker using commercial equipment (e.g., one to several weeks) and a low effort for a decently funded attacker using distributed computing or optimized hardware (e.g., one to several days).

Impact. The BLUFFS attacks have a *severe* impact on Bluetooth's security and privacy. They allow decrypting (sensitive) traffic and injecting authorized messages across sessions by re-using a *single* session key. Prior attacks require leaking *PK* or brute-forcing one *SK* per target session to achieve a similar impact. Our attacks can target any Bluetooth device, regardless of its role, security mode, and supported Bluetooth profiles, as they rely on flaws in the standard (detailed next in Section 4.2). Moreover, the attacks are *stealthy* since they exploit the Bluetooth firmware (Controller) without requiring user interaction and triggering notifications to the user. Finally, the attacks do *not* require specialized and expensive equipment, as demonstrated by our implementation in Section 5, and have a *widespread impact*, as empirically shown in Section 6. Motivated by the impact of the attacks, in Section 7, we present a practical design-level fix that we recommend integrating into the Bluetooth standard and discuss other implementation-level mitigations.

4.2 Attacks Root Causes

The BLUFFS attacks' root causes are *four* architectural vulnerabilities in the specification of Bluetooth session establishment (i.e., RC1, RC2, RC3, and RC4) [11]. RC1 and RC2 are *novel* as they are the first targeting *SK* derivation and allowing to derive the same *SK* across sessions (breaking their forward and future secrecy). While RC3 and RC4 have been exploited to attack other session establishment phases. For instance, the BIAS attacks [3] employ RC3 and RC4 to bypass *PK* authentication, while the KNOB attacks [5] take advantage of them to downgrade the entropy of *SK*.

Table 1: Mapping the six BLUFFS attacks to their four root causes. CI and PI stands for Central Impersonation and Peripheral Impersonation.

BLUFFS attack	RC1	RC2	RC3	RC4
A1: Spoofing a LSC Central	✓	✓	✓	×
A2: Spoofing a LSC Peripheral	✓	✓	✓	×
A3: MitM LSC victims	✓	✓	✓	×
A4: Spoofing a SC Central	✓	✓	✓	✓
A5: Spoofing a SC Peripheral	✓	✓	✓	✓
A6: MitM SC victims	✓	✓	✓	✓

RC1: LSC SK diversification is unilateral (new). The LSC SKDF introduced in Section 2 and depicted in Figure 1 derives a *SK* using static inputs (i.e., *PK*, *BA*) and variable ones (i.e., *AC*, *SE*, *SD*). The variable inputs diversify *SK*s across sessions. One would expect that both the Central and the Peripheral would contribute to *SK* diversification. However, the standard allows the *Central* to set all the *SK* diversification values. Hence, an attacker impersonating a Central (or role switching to a Central when impersonating a Peripheral) can *unilaterally drive SK* diversification (across sessions). We note that the Peripheral's Bluetooth address is unusable as a variable input because Bluetooth (Classic) does *not* support randomized link-layer addresses.

RC2: LSC SK diversification does not use nonces (new). *SK* is diversified using random numbers (*AC* [11, p. 625] and *SD* [11, p. 637]) and a positive integer (*SE* in [11, p. 962]). As none of them is a nonce, they can be *reused* in past, present, and future sessions without violating the standard. Hence, an attacker who knows a triplet (AC_C, SE_C, SD_C) and the corresponding SK_C, can force the victims to derive the same *attacker-controlled* session key across sessions.

RC3: LSC SK diversifiers are not integrity protected. The variable inputs exchanged during *SK* derivation are sent without integrity protection. As a result, an attacker who is spoofing a device or performing MitM on a session can manipulate *AC*, *SE*, and *SD*, without being detected.

RC4: Downgrading SC to LSC does not require authentication. The negotiation of SC or LSC is not integrity protected. Hence, an attacker can always downgrade a session to LSC, regardless of SC support from the victim, and trigger LSC key negotiation and KDF_{LSC} (presented in Figure 1).

Root causes and attacks. Table 1 shows how the six BLUFFS attack presented in Section 4.1 map to RC1, RC2, RC3, and RC4. All attacks take advantage of RC1, RC2, and RC3 as they unilaterally derive a constant session key without using a nonce and manipulating the integrity of the session key diversifiers. RC4 is exploited by the three BLUFFS attacks targeting SC to downgrade a session to LSC. We also note that no prior research (and attack) discovered RC1 and RC2.

4.3 Comparison with KNOB and BIAS

The KNOB+BIAS attack chain is considered the most effective way to impersonate Bluetooth devices during session establishment. The attacker employs BIAS to bypass *PK*'s authentication, then KNOB to downgrade the entropy of *SK*. The BLUFFS attacks share the same goals but employ *different* steps (e.g., attacking *SK* derivation) that are *chainable* with the BIAS and KNOB ones.

However, unlike the BLUFFS attacks, the KNOB+BIAS chain does *not* compromise forward and future secrecy as it is effective within the current session. More generally, no prior research investigated the existence of vulnerabilities and attacks on session establishment persisting *across* sessions (i.e., no research on Bluetooth sessions' forward and future secrecy). Our work fills this research gap by presenting the first key-reuse attacks for Bluetooth.

The BLUFFS attacks are successful even if we fix the role-switching and SC session downgrade vulnerabilities discussed in the BIAS paper. The attacker can reuse SK_C against any LSC device while impersonating an LSC Central (A1). In particular, the attacker legitimately negotiates LSC, AC_C, SE_C, SD_C and is not required to authenticate *PK*. Moreover, devices patched against the KNOB attacks are still vulnerable to the BLUFFS attacks, as they accept SE_C equal to seven.

We enable attack scenarios, which are too costly for KNOB+BIAS. For instance, if we target N_s sessions, our attacks' cost does *not* increase with N_s as we brute force one session key. While the KNOB+BIAS cost is significantly higher as it *linearly* increases with N_s. The cost difference is even more compelling if a victim supports entropy values (*SE*) higher than seven bytes. To give an intuition about the cost difference, if we assume that brute forcing a *SK* with seven bytes of entropy takes one week (keyspace is 2^{56}), and a *SK* with sixteen bytes of entropy takes one thousand years (keyspace is 2^{128}); then our attacks cost one week against seven bytes of entropy and one thousand years against sixteen bytes of entropy, while KNOB+BIAS costs N_s weeks and N_s thousand years.

As a result of our investigation, we formulate and empirically answer new and valuable research questions not addressed by KNOB and BIAS (and any other prior work). For example, we reveal the forward and future secrecy guarantees provided by the Bluetooth standard, their architectural vulnerabilities, how to exploit these vulnerabilities with practical and low-cost attacks, the attacks' effectiveness on actual devices from different hardware and software providers, and how to fix or mitigate the attacks (and their root causes).

5 IMPLEMENTATION

We now describe the implementation of our `BLUFFS` toolkit to perform and check the BLUFFS attacks presented in Section 4. The toolkit has two modules: an *attack device* and an *attack checker* and extends state-of-the-art tools for Bluetooth security research, such as `internalblue` [47] and the `BIAS` and `KNOB` toolkits [17, 18], with novel and useful features. For example, we unlock the possibility to dynamically manipulate Bluetooth's key derivation parameters and monitor *SK* across sessions, and automatically detect our attacks from a pcap file.

Our toolkit is *low-cost* as it uses open-source software (e.g., Python and Wireshark) and cheap hardware (e.g., a Linux laptop and

a Cypress CYW20819 development board). Its technical details are relevant for reproducing, checking, and extending our experimental setup and results (shown in Section 6). We are anonymously releasing the attack checker module and selected parts of the attack device module at https://anonymous.4open.science/r/sec23-anon-654A.

5.1 Attack device module

Architecture. Our attack device consists of a *Linux laptop* connected via USB to a *CYW20819 board* from Cypress/Infineon. Its initialization setup is the same as the one described in the BIAS repository [18]. In summary, to access link-layer traffic from the laptop's HCI interface, we activate LMP redirection from the board with a vendor-specific command and patch the laptop's Linux kernel to parse the LMP packets. Moreover, we patch the board's firmware using a proprietary binary instrumentation feature from Cypress. Patching the firmware (Bluetooth Controller) is essential to manipulate Bluetooth key derivation. The board's patching is facilitated by *Internalblue* [39], which provides high-level Python APIs to patch the board (i.e., `patchRom`) and read and write its RAM (i.e., `readMem` and `writeMem`).

The CYW20819's vendor-specific patching mechanism is quite complex but clever. First, the unpatched firmware, stored in read-only memory (ROM), receives the `Download_Minidriver` command from our laptop (Bluetooth Host) and stops its execution. Then, the laptop sends a `Write_RAM` command to write in RAM the addresses to be modified in ROM. Finally, the laptop runs the `Launch_RAM` command to register the patches in RAM and resume execution. Hence, anytime the firmware CPU fetches an address in ROM that should be patched, the control flow is redirected to the patch in RAM. For more information about this mechanism, refer to [33].

Firmware patches. We developed *seven new patches* for the attack device Bluetooth firmware. The patches, summarized in Table 2, allow performing the six BLUFFS attacks presented in Section 4. The table's first and second columns indicate the patch name and description, while the last two show the patched firmware function and its ROM address.

Our patches unlock useful security testing capabilities for Bluetooth. The three *man_** patches manipulate *AC*, *CR*, and *SD*, and enable negotiating constant *SK* diversifiers as in Figure 2, and failing session establishment when the attacker has to authenticate a *PK*. The three *rea_** patches monitor *SK*'s value which is otherwise *hidden* to the HCI and LMP layers. The *rs_nop* patch allows to successfully attack devices asking to role switch to Central regardless of the attacker's role switch strategy. This patch is valuable as it extends the effectiveness of our attacks (and the BIAS+KNOB chain) to a new class of devices. We reuse the patches from the `BIAS` toolkit [18] to negotiate $SE = 7$ for the KNOB attack and avoid *PK* authentication. We also coded a high-level patching function to ease the development of new patches (see `device/patch.py` in our anonymized repository).

We developed the patches in Table 2 by *reverse-engineering (RE)* unknown portions of the CYW20819 Bluetooth firmware. In particular, we used Ghidra [55] loaded with the firmware symbols leaked from a Cypress SDK as described in [39]. As we wrote the patches in ARM Thumb-2 assembly, they contain 2-byte and 4-byte instructions aligned to 4-byte boundaries, and the code branches to odd addresses [6]. Currently, to comply with responsible disclosure, we are releasing `man_cr.s`, `rea_sk.s`, and `rs_nop.s`.

Table 2: Seven novel patches for the CYW20819 Bluetooth firmware to perform the BLUFFS attacks. The third and fourth columns indicate the patched firmware function and its address in ROM.

Name	Description	Patched function	Addr
man_ac	Manip. *AC*	`txAuRand`	`AEB8C`
man_cr	Manip. LSC *CR*	`txSres`	`AEDC8`
man_sd	Manip. *SD*	`txStartEncryptReq`	`AE4B4`
rea_sk	Read *SK* value	`txStartEncryptReq`	`AE5B4`
rea_skec	Read Central *SE*	`txStartEncryptReq`	`AE5B4`
rea_skep	Read Perip. *SE*	`procStartEncryptReq`	`AE70C`
rs_nop	No Perip. role sw.	`handleLmpSwitchReq`	`A643C`

Listing 1 shows our *rs_nop* patch to refuse Peripheral's role switch requests. Whenever the firmware program counter hits `0xA643C` inside `handleLmpSwitchReq` in ROM, the firmware code jumps to our patch in RAM. The patch passes a zero as `isMssInstantPassed`'s second parameter by zeroing `r1`. Then, it calls (i.e., branch and link) `isMssInstantPassed` and overwrites the routine's return value to `True` by setting `r0` to one. As a side effect, the attack device firmware thinks that the MSS (Minimum Subevent Space) interval has passed and *rejects* the correspondent role switch request. Notably, such rejection is compliant with the standard. Finally, the patch unconditionally jumps back to the next valid ROM instruction in Thumb-2 mode (i.e., branch to an odd address). This patch enables exploitation of a *new* class of devices, such as victims trying to (defensively) role switch to the Central role during session establishment. For example, we can exploit iPhone 12 and 13 by rejecting their role switch requests during session establishment.

5.2 Attack checker module

Our attack checker enables new capabilities for Bluetooth static analysis. In particular, given a pcap file containing LMP packets, it automatically isolates Bluetooth sessions, computes session keys, and detects the BLUFFS attacks. We release it as part of our `BLUFFS` toolkit in the `checker` folder. The checker is written in Python 3 and leverages capable and available tools, such as `wireshark`/`tshark` [59] and `pyshark` [20]. It requires H4 and LMP dissectors for Wireshark v3.6+ [48] or older versions [19]. We now describe the checker's *parser*, *kdf*, and *analyzer* components.

Parser. The parser uses `pyshark` to extract relevant LMP packets from a pcap file. It supports *nine* LMP packet types as shown in Table 3. Specifically, it parses `LMP_host_connection_req` and `LMP_detach` packets, which indicate when a session starts and ends. It processes entropy negotiation values (*SE*) from `LMP_encryption_key_size_req` and related `LMP_accepted` packets. The parser also manages authentication challenges (*AC*) and responses (*CR*) from `LMP_au_rand` and `LMP_sres` packets and detects when *AC* is not accepted by monitoring the relevant `LMP_not_accepted`

Table 3: Nine LMP packets supported by our parser.

LMP packet	Opcode	Description
`LMP_host_connection_req`	51	Start a session
`LMP_detach`	7	Abort a session
`LMP_encryption_key_size_req`	16	Propose *SE*
`LMP_accepted` (*SE*)	3 (16)	Confirm *SE*
`LMP_au_rand`	11	Send *AC*
`LMP_sres`	12	Send *CR*
`LMP_start_encryption_req`	17	Send *SD*
`LMP_accepted` (*SD*)	3 (17)	Accept *SD*
`LMP_not_accepted` (*AC*)	4 (11)	Reject *AC*

packet. Moreover, it deals with session key diversifies (*SD*) by parsing `LMP_start_encryption_req` and consequent `LMP_accepted` packets.

The parser's implementation is at `device/parser.py` and follows an *object-oriented* design. An `LmpBase` parent class, shown in Listing 2, parses relevant fields shared by *all* LMP packets. For example, it stores the LMP packet number (`number`), transaction initiator (`tinit`), and opcodes (`op`, `op_ext`). Specialized classes, extending `LmpBase`, manage specific LMP opcodes. For instance, `LmpAuRand`, presented in Listing 3, deals with `LMP_au_rand` packets and extracts *AC* as an hexstring and a bytearray (`aurand` and `aurand_ba`). We developed other eight specialized LMP classes, see `parser.py` for more details.

Kdf. The kdf module implements the LSC key derivation function presented in Section 2 as shown in Listing 4. This functionality is needed to compute and check *SK*s across sessions automatically. In particular, `kdf.py` computes *SK* (as in Equation 1) by using `e1.py`, `e3.py` and `es.py` and their related cryptographic primitives (such as `h.py`). We provide the kdf code in the toolkit's `device` folder, and we note that it extends [17, 18] by providing the *full* LSC key derivation chain. Our code is sound as is *tested* against the vectors in the Bluetooth standard [11, p. 921] and actual values extracted during our experiments. The kdf test suite can be run with `make tests`.

Analyzer. The analyzer module is implemented in `checker/analyzer.py` and *automatically* detects the BLUFFS attacks presented in Section 4. It builds on top of the parser and kdf modules presented earlier. The analyzer employs the `gen_analysis` function, shown in Listing 5, that takes as inputs a pcap file, a *PK*, and the Bluetooth address of the victim (Peripheral). Then it calls `gen_sessions` to extract from the pcap a list of LMP sessions (`sessions`). Then, for each session, it calls the `gen_report` function that computes *SK* from *SE*, *SD*, and *AC* and stores the reports in a list (`reports`). Finally, for each report `gen_analysis` checks if SK_C is reused across sessions (`assert report["sk"] == EXP_SK`). This automation speeded up our large-scale evaluation reported in Section 6.

To demonstrate that our module is practical, we provide the material to reproduce our analysis of the Pixel Buds A-Series earbuds. In the toolkit's `pcap` folder, there is a file prefixed with `lsc-` with the LMP traffic generated while we performed the PI and CI attacks while spoofing an LSC device. Also, we provide a `sc-` prefixed file for the CI and PI attacks while impersonating a SC device. `analyzer.py` contains two test functions with the needed *PK*, *BA*, and target SK_C. By running the script, we observe that the attacker *reuses* SK_C across sessions, regardless of her role (i.e., Central or Peripheral). In particular, in the LSC cases SK_C is `c61da2f42fefab75bb15b7927af0a631`, while in the SC scenarios is `3581f68eecc5d1f295894c6bc9262812` and both SK_C have 7 byte of entropy. Under the hood, the script verifies SK_C (`EXP_SK`) with an `assert` statement at line 175. The first session in each test contains an *SK* different from SK_C, as that session is *not* under attack, but it is the first legitimate session after pairing completion.

6 EVALUATION

We now present our evaluation setup and results.

6.1 Setup

Our evaluation setup tests the six BLUFFS attacks presented in Section 4 (i.e., A1, A2, A3, A4, A5, and A6) on a target LSC or SC device. Testing a device requires less than 15 minutes. Our setup relies on the attack device and checker modules introduced in Section 5 to automate its repetitive parts (e.g., compute and check the session keys from a pcap file). The setup has *six* steps:

(1) First, we test A4, A5, and A6 which involve spoofing and MitM of SC victims. We pair the attack device (also acting as a spoofed victim) with the target victim, and we disconnect them. While pairing, the attack device declares *SC support*.
(2) We patch the attack device's firmware (using the patches presented in Table 2) to implement the strategy discussed in Section 4.1. The patched attack device declares LSC support, monitors *SK*s across sessions, and sets $AC_C = SD_C = 0$, and $SE_C = 7$ to renegotiate a constant and weak session key (i.e., SK_C). Also, the attack device tries to role switch to Central before session key derivation when it is a Peripheral and refuses role switch requests when acting as a Central. We also force the attack device to send a wrong CR_C to detect a failure in (rare) attack scenarios where the victim asks the Central to authenticate *PK* (e.g., PI against the BOOM 3 Bluetooth speaker).
(3) We test A4 by establishing multiple sessions from the attack device (Central) and capturing the HCI and LMP packets in a pcap file. We also monitor *SK* from RAM in each session, but this manual step is optional. Then, we employ our attack checker to automatically recompute and compare the *SK*s from the pcap file. If the computed keys are the same, the attack is successful, as the adversary is impersonating a SC device while reusing SK_C across sessions.
(4) We test A5 by establishing multiple connections from the victim to the attack device (Peripheral). We employ the same strategy described in the previous steps, and the attack is effective if we reuse SK_C across sessions.
(5) If either the CI or the PI attack is successful, then the victim is also vulnerable to A6, as the adversary can combine CI and PI in a MitM attack against SC victims.
(6) We unpair the attack device and the victim and pair them again, but this time the attack device declares *LSC support*.

Then, we repeat steps two, three, four, and five to test A1, A2, and A3.

Our setup uses, *without loss of generality*, the attack device both as a victim and the attacker to speed up the experiments. However, as stated in Section 3, we stress that the attacker does require neither to pair with the victim devices nor observe them while they are pairing nor trigger a new pairing session. To prove such a claim, we tested scenarios where before attacking the victim, we unpaired the attack device from the victim by overwriting its *PK* with a wrong value (via a firmware patch), and we were still able to force SK_C across sessions.

6.2 Results

Table 4 presents our evaluation results obtained by testing the six BLUFFS attacks on *eighteen* heterogeneous and popular devices (second column) embedding *seventeen* unique Bluetooth chips (first column) and employing the most popular Bluetooth versions (third column). We compiled the table following the six steps in Section 6.1. The last six columns contain a ✓ if a device is vulnerable to an attack; otherwise, a ×. The fourth, fifth, and sixth columns show CI, PI, and MitM attacks when the spoofed victim supports LSC (i.e., A1, A2, and A3). While the last three columns report CI, PI, and MitM attacks while impersonating a SC device (i.e., A4, A5, and A6).

LSC Victims. As shown by the first six rows in Table 4, *all* tested LSC chips and devices are vulnerable to the six attacks, with one exception. The Logitech BOOM 3 speaker is *not* vulnerable to the PI attacks (A2, A5), as it requires the Central to authenticate *PK*, thus preventing the attacker from completing session establishment (despite eventually being able to reuse SK_C). The Bose SoundLink speaker also asks the Central to authenticate but is still vulnerable to A2 and A5 as it does *not* check the challenge response. The Google Pixel Buds A-Series (2021) are still vulnerable to the KNOB downgrade resulting in SK_C with *1 byte* of entropy; we reported this worrisome finding to Google and got a "will not fix" response.

SC Victims. The last *eleven* rows in Table 4 shows our findings about chips and devices supporting *SC*. If the spoofed victim supports LSC, all chips/devices are vulnerable to the CI, PI, and MitM attacks (A1, A2, A3). Hence, an attacker can impersonate any chip/device from the LSC block of rows to any chip/device in the SC set. If we impersonate a SC device, the CYW20819 and CYW40707 chips are vulnerable to A4, A5, and A6, demonstrating that the attacks are effective against SC. Instead, the other eight chips/devices we tested are *not* vulnerable to A4, A5, and A6, as the chips enforce SC between pairing and session establishment, preventing the attacker from downgrading the session to LSC. But, they are still vulnerable to A1, A2, and A3 because of the vulnerabilities we uncover with LSC.

Evaluation impact. Driven by our empirical results shown in Table 4, we are convinced that the BLUFFS attacks are *practical* and have a *large-scale* impact on the Bluetooth ecosystem. In particular, they can target *SC and LSC* devices (e.g., laptops, smartphones, headsets, and speakers) supporting a wide range of *operating systems* (e.g., iOS, Android, Linux, Windows, and proprietary OS), *Bluetooth stacks* (e.g., BlueZ, Gabeldorsche, Bluedroid, and proprietary ones), *vendors* (e.g., Intel, Broadcom, Cypress, Cambridge Silicon Radio, Infineon, Bestechnic, Apple, Murata, Universal Scientific Industrial, Samsung, Dell, Google, Bose, Logitech, Xiaomi, Lenovo, Jaybird, and Qualcomm), and *Bluetooth versions* (e.g., 5.2, 5.1, 5.0, 4.2, and 4.1).

Moreover, Table 4's list of vulnerable chips and devices represents a *lower bound*. We cannot test all Bluetooth devices in the market. However, we are confident that most of them are flawed, as the BLUFFS attacks exploit architectural issues of Bluetooth session key derivation. We can confidently infer that all untested devices employing an exploitable chip from Table 4 are vulnerable. For instance, since the Apple H1 chip is in our list; we can predict that the other devices embedding H1 are also affected, e.g., AirPods gen. 2 and 3, AirPods Max, Beats Solo Pro, Powerbeats (2000), Powerbeats Pro, and Beats Fit Pro [58]. Hence, there is a need for a usable countermeasure to fix the BLUFFS attacks by-design, and we address this challenge in Section 7.

7 ENHANCED LSC KDF

Motivated by the impact of our attacks (e.g., results from Section 6.2), we present an *enhanced LSC KDF* addressing the six BLUFFS attacks and their four root causes at the architectural level. Our KDF uses authenticated and mutual key derivation and is backward compliant with KDF_{LSC} (Section 7.1). We show how to integrate our countermeasure in the Bluetooth standard while entailing minimal computation, throughput, and latency overheads (Section 7.2). The fix also aligns with best practices for symmetric key derivation, such as NIST SP 800–56C–rev2 [7]. We report how we successfully tested our fix at the protocol level. Based on our results, we recommend its introduction in the Bluetooth specification (e.g., via an amendment). We also discuss low-cost *implementation-level* mitigations that vendors can employ until the standard is updated (Section 7.4).

7.1 Design

Figure 4 shows the *message sequence chart* of our enhanced KDF which extends KDF_{LSC}, described in Figure 1, in four ways:

(1) Adds EKD, a feature flag to negotiate our KDF, as shown by the first two messages in Figure 4. This flag provides *backward compatibility* as it accommodates devices supporting and not supporting our protocol. It can also enforce the usage of our protocol across sessions, avoiding (malicious) KDF downgrades. The Bluetooth standard employed the same approach when it introduced SC.
(2) Defines *SD* not as a random number but as a *nonce*, (i,e., number usable once). This definition is valuable as it mandates by design to deny *SD*'s re-usage, regardless of the attacker's strategy.
(3) Employs the *mutually authenticated key diversification scheme* presented in Figure 4, rather than the unilateral and unauthenticated one from the standard. In particular, Alice sends Bob SD_A (i.e., Central *SD* nonce) and Bob answers with $Mac(SD_A, PK)$, a message authentication code (MAC) computed from the diversifier and *PK* to acknowledge and authenticate it. Alice aborts the session if the MAC check fails while Charlie cannot produce such MAC since she does not

Table 4: BLUFFS attacks evaluation results. We run the six BLUFFS attacks against *eighteen* devices with *seventeen* unique Bluetooth chips. All the six tested LSC victims are vulnerable to all the attacks, with one exception. When we impersonate an LSC device to an SC device, all tested eleven SC targets are vulnerable. Comparatively, when we spoof an SC device to another SC device, the attacks are only effective on *two* out of eleven tested chips (i.e., CYW20819 and CYW40707). Our results empirically demonstrate that the attacks are practical and have a widespread impact on the Bluetooth ecosystem. Notes: [1]ask to authenticate as a Central, [2]does not check authentication response (*CR*), [3]vulnerable *SK* downgrade with 1 byte of entropy, [4]does not allow LSC session establishment if paired with SC. Acronyms: USI stands for Universal Scientific Industrial, CYW for Cypress, BCM for Broadcom, and CSR for Cambridge Silicon Radio. A n/a in the Chip column indicates that the chip SoC model is unavailable from public sources.

Chip	Device(s)	BTv	A1	A2	A3	A4	A5	A6
LSC Victims								
Bestechnic BES2300	Pixel Buds A-Series[3]	5.2	✓	✓	✓	✓	✓	✓
Apple H1	AirPods Pro	5.0	✓	✓	✓	✓	✓	✓
Cypress CYW20721	Jaybird Vista	5.0	✓	✓	✓	✓	✓	✓
CSR/Qualcomm BC57H687C-GITM-E4	Bose SoundLink[1,2]	4.2	✓	✓	✓	✓	✓	✓
Intel Wireless 7265 (rev 59)	Thinkpad X1 3rd gen	4.2	✓	✓	✓	✓	✓	✓
CSR n/a	Logitech BOOM 3[1]	4.2	✓	×	✓	✓	×	✓
SC Victims								
Infineon CYW20819	CYW920819EVB-02	5.0	✓	✓	✓	✓	✓	✓
Cypress CYW40707	Logitech MEGABLAST	4.2	✓	✓	✓	✓	✓	✓
Qualcomm Snapdragon 865	Mi 10T[4]	5.2	✓	✓	✓	×	×	×
Apple/USI 339S00761	iPhones 12[4], 13[4]	5.2	✓	✓	✓	×	×	×
Intel AX201	Portege X30-C[4]	5.2	✓	✓	✓	×	×	×
Broadcom BCM4389	Pixel 6[4]	5.2	✓	✓	✓	×	×	×
Intel 9460/9560	Latitude 5400[4]	5.0	✓	✓	✓	×	×	×
Qualcomm Snapdragon 835	Pixel 2[4]	5.0	✓	✓	✓	×	×	×
Murata 339S00199	iPhone 7[4]	4.2	✓	✓	✓	×	×	×
Qualcomm Snapdragon 821	Pixel XL[4]	4.2	✓	✓	✓	×	×	×
Qualcomm Snapdragon 410	Galaxy J5[4]	4.1	✓	✓	✓	×	×	×

know *PK*. Then, the protocol enforces a similar exchange of messages from Bob to Alice involving SD_B (i.e., Peripheral *SD* nonce) and $Mac(SD_B, PK)$. After exchanging these messages, Alice and Bob mutually set and authenticate the session key diversifiers.

(4) Uses the $MKDF_{LSC}$ *mutual key derivation function* to compute mutually diversified *SK*, unlike KDF_{LSC} that allows a single (malicious) party to diversify *SK*. In particular, $MKDF_{LSC}$ binds *SK* to SD_A and SD_B, the authenticated nonces sent by Alice and Bob.

Our enhanced KDF fixes the four attack root causes presented in Section 4.2. RC1: The key diversification is *mutual* as *SK* depends on contributions from the Central and the Peripheral (i.e., SD_A and SD_B). RC2: The diversifiers are defined as *nonces* rather than random numbers. RC3: The negotiation of the diversifiers is *integrity protected* using message authentication codes keyed with *PK*. RC4: We tolerate (malicious) LSC to SC downgrades by providing a stronger LSC key derivation protocol.

Our scheme stops the six BLUFFS attacks regardless of the attacker's role (CI, PI, or MitM) and target security mode (LSC or SC). In particular, the attack strategy presented in Figure 2 becomes ineffective, as the victim asks the other party to authenticate *SD* with *PK* and aborts session establishment if authentication fails. Furthermore, the fix prevents the attacks *even if* the attacker successfully authenticates (e.g., by stealing *PK*), as the attacker cannot control the victim's *SD* to force a known *SK*.

Despite being designed to address the BLUFFS vulnerabilities, our KDF mitigates the KNOB attacks and stops the BIAS attacks. The KDF increases the *SK* brute force effort *exponentially* with the negotiated entropy and *linearly* with the number of target sessions as the attacker must brute force a *new SK* for each session, other than a single *SK* regardless of the number of target sessions. Hence, our KDF is effective even if the attacker can brute force SK_C as her effort to target n sessions increases from 2^{56} to $n \times 2^{56}$. Moreover, it blocks the BIAS attacks as an adversary who managed to skip *PK* authentication (e.g., by attacking a victim not patched against BIAS) cannot bypass our mutually authenticated key derivation protocol without knowing *PK*.

7.2 Integration in the Bluetooth Specification

Our fix requires backward-compliant modifications to the Bluetooth standard (e.g., LSC session establishment) and produces minimal overhead (e.g., one extra negotiation bit, three extra LMP packets carrying in total 48 bytes of extra data to authenticate the diversifiers, three extra function calls to compute the MACs and *SK*). We now describe these modifications in detail.

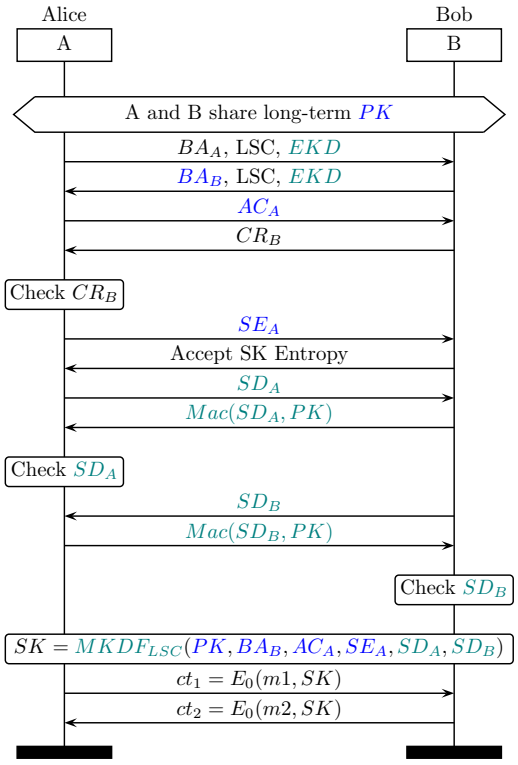

Figure 4: Enhanced LSC session key derivation. Alice and Bob negotiate the enhanced KDF via EKD to maintain backward compatibility with KDF_{LSC}. They mutually exchange, authenticate, and check session key diversification nonces (i.e., SD_A, SD_B, $Mac(SD_A, PK)$, $Mac(SD_B, PK)$). Then, use the diversifiers in a mutual key derivation function (i.e., $MKDF_{LSC}$) to compute fresh and non-reusable keys across sessions. Our enhanced KDF fixes the BLUFFS attacks and their four root causes by design.

EKD requires adding a new LMP feature that should be stored in the firmware and optionally in the OS. For instance, the standard could introduce an EKD flag usable to negotiate our enhanced KDF during LMP feature exchange (as in Figure 4). Moreover, a device can enforce EKD usage among sessions and refuse to connect with a device not supporting it.

Mandating nonces rather than random looking SDs requires straightforward textual modification to the standard. For example, instead of defining SD as EN_RAND [11, p. 637], the standard should define it as EN_NONCE. Or when talking about SDs, the document should classify them as "nonces" other than "random numbers".

Authenticating SD is also easy to implement as during session establishment Alice and Bob already share PK. In particular, we recommend computing the MACs reusing the e_1 authentication function from the standard [11, p. 975] as follows:

$$Mac(SD_A, PK) = e_1(PK, SD_A, BA_B) \quad (2a)$$

$$Mac(SD_B, PK) = e_1(PK, SD_B, BA_A) \quad (2b)$$

$MKDF_{LSC}$ is a backward compatible extension of KDF_{LSC} presented in Equation 1. COF and ISK are computed as in KDF_{LSC} (i.e., Equations 3a and 3b). Then, we add Equation 3c to bind the session key to SD_B by computing a second intermediate session key ISK', reusing the e_3 key generation function [11, p. 981]. In Equation 3d, we reuse e_s to reduce the session key entropy as usual and produce SK. In summary, $MKDF_{LSC}$ is described by the following four equations:

$$COF = e_1(PK, AC_A, BA_B) \quad (3a)$$

$$ISK = e_3(PK, SD_A, COF) \quad (3b)$$

$$ISK' = e_3(ISK, SD_B, COF) \quad (3c)$$

$$SK = e_s(ISK', SE_A) \quad (3d)$$

Lastly, we propose two extensions of the LMP protocol to mutually generate and authenticate SD. First, the `LMP_start_encryption_req` command (opcode 17) which now is used to send SD from the Central [11, p. 638], should be usable also by the *Peripheral* to send its diversifier nonce. Second, we require a *new* LMP command, defined as `LMP_start_encryption_res`, to send a 16 Byte MAC authenticating an SD. Indeed, if Alice is the Central and Bob the Peripheral, we expect the following four LMP messages:

(1) Alice: LMP_start_encryption_req(SD_A)
(2) Bob: LMP_start_encryption_res($Mac(SD_A, PK)$)
(3) Bob: LMP_start_encryption_req(SD_B)
(4) Alice: LMP_start_encryption_res($Mac(SD_B, PK)$)

7.3 Protocol Level Evaluation

The `BLUFFS` toolkit includes a Python implementation of our enhanced LSC session key derivation (see `checker/mkdf.py`). We used our implementation to empirically confirm at the *protocol-level* that the BLUFFS attacks are not effective by testing the same attack scenarios exploited in Section 4 using the attack strategy in Figure 2. Hence, the presented KDF stops the attacks and their root causes (i.e., exploited vulnerabilities) by design.

As shown in `checker/mkdf_tests.py`, the attacker controls AC_C (`AU_RAND_C`), SD_C (`EN_NONCE_C`), SE_C (`ENTROPY_C`). However, she *cannot* authenticate the victim's session key diversifier (`MAC_V`) as she does not know PK (`LK`). Even if the adversary manages to bypass SD mutual authentication, she cannot force a known SK_C as she does not control SD_V (`EN_NONCE_V`). As a result, the attacker cannot conduct the BLUFFS attacks, regardless of her role (Central or Peripheral) and the type of spoofed victim (LSC or SC).

7.4 Implementation Level Mitigations

SC-to-SC enforcement. Enforcing SC mode between pairing and session establishment stops the attacks when both victims support SC. One can implement this enforcement in the OS (i.e., Bluetooth Host) by storing a SC flag for each paired device and checking that flag during session establishment. As a result, if the attacker impersonates a SC device, the victim can check whether or not the impersonated device supports SC and abort the session when the attacker negotiates LSC. From Table 4 – Note 4, we see that ten SC devices already implement this fix. Unfortunately, this mitigation only covers SC-to-SC attack scenarios that currently are less prevalent than LSC-to-SC and LSC-to-LSC ones.

LSC SD cache. A device can stop the presented attacks by maintaining a cache of seen *SD* (i.e., LSC session key diversifiers) and refusing a connection with a (malicious) device proposing a *SD* in the cache. One can implement this cache in the Bluetooth firmware (i.e., Bluetooth Controller), as *SD* is not visible by the OS. Unfortunately, this mitigation could be brittle as the cache is unauthenticated. For instance, an adversary can poison the cache with dumb *SD*s and then negotiate the target *SD*, which is no more in the cache.

LSC Central authentication. A device can stop the PI attacks by requiring an attacker in the Central role to authenticate *PK*. One can implement this check in the Bluetooth firmware by updating the session establishment code in a backward compliant way. As a result, the attacker cannot complete LSC session establishment, as she cannot authenticate *PK*. This mitigation is implemented by the Logitech BOOM 3 speaker, as shown in Table 4 and detailed in Note 1. Notably, Central authentication only protects against the two PI attacks.

8 RELATED WORK

Attacks on Bluetooth session establishment. Our work is the first presenting attacks breaking Bluetooth's forward and future secrecy and persisting across sessions. Other attacks on session establishment showed that session entropy negotiation is vulnerable to downgrade attacks reducing the strength of *SK* to 1 byte [5]. The standard now mandates a minimum entropy value of 7 bytes, but recent work showed that some devices classes still accept 1 byte of entropy [2]. Other work uncovered how to bypass session authentication [3, 36]. Recent work analyzed how to employ KNOB+BIAS to exploit different Bluetooth profiles within the same session [1] and the vehicular ecosystem [2].

Attacks on Bluetooth pairing. Several works targeted Bluetooth pairing (i.e., the SSP protocol), while in this work, we assume that it is not under attack. In particular, there are SSP's probabilistic invalid curve attacks [9], MitM attacks [25, 53], and cross-transport key derivation attacks [4]. Moreover, related work targeted SSP association [29, 56] and the legacy pairing protocol [32, 37, 51]. We note that attacks on pairing are more challenging to perform and less stealthy than ones on session establishment as pairing is a one-time procedure involving user interaction.

Bluetooth tracking attacks. The fact that Bluetooth addresses are not randomizable not only helps to perform the BLUFFS attacks but also enables device tracking threat where an adversary violates the victim's privacy by tracking his movements using the Bluetooth address as a permanent identifier [16, 26, 27, 60]. Researchers proposed similar attacks for BLE, despite its usage of allowlists and address randomization [61].

Bluetooth firmware research. Bluetooth firmware are essential for security research as they implement pairing and session establishment. However, they are proprietary and closed-source, requiring significant reverse-engineering effort to be analyzed and patched. Luckily, researchers have developed tools to inspect and patch popular Bluetooth firmware. For example, Internalblue [39] provides a Python API to interact and patch popular Broadcom and Cypress firmware. Other work focused on Bluetooth firmware's automated extraction of security-related parameters [52], detection of link-layer vulnerabilities [57] and weaknesses in random number generation [54].

Bluetooth fuzzing and implementation bugs. We discovered the protocol-level BLUFFS attacks and their root causes by inference from the Bluetooth specification. Then, we automated the repetitive tasks by developing a toolkit. Other research work used directed fuzzing to find crashes, denial of service (DoS), and remote code execution (RCE) *implementation-level* bugs in popular Bluetooth stacks [24, 44, 46]. Other recent work employs differential testing to catch protocol compliance implementation bugs [34] automatically. There are works uncovering implementation-level vulnerabilities resulting in RCE using semi-automated techniques. Notable examples are: BlueBorne [49] impacting Amazon Echo and Google Home, BlueFrag [23] against Android 9, Bleedingbit [50] on Texas Instrument BLE chips, and BleedingTooth [42] targeting BlueZ and the Linux kernel. Unlike the exploits described in this paragraph, the BLUFFS attacks can target a device *regardless of* its implementation details (and bugs).

Forward/future secrecy. Forward and future secrecy were extensively studied for Transport Layer Security (TLS) and Instant Messengers (IM). TLS's forward secrecy was evaluated in the wild [28] and TLS 1.3 mandates it using non-static cipher suites, such as ephemeral Diffie-Hellman (DH) key exchange [31]. The double ratchet algorithm [45], used by the most popular IMs (e.g., Signal, and WhatsApp), provides future secrecy with the DH ratchet and forward secrecy with the symmetric ratchets and was analyzed by security researchers [15]. No prior work evaluated Bluetooth's forward and future secrecy properties (not even the Bluetooth standard).

Survey on Bluetooth security. There are not so recent survey papers about Bluetooth security [21, 38, 41, 43]. They are an excellent way to get introduced to Bluetooth's security architecture and related threats. However, none of them discusses Bluetooth's forward and future secrecy guarantees.

9 CONCLUSION

This paper presents the first security evaluation of Bluetooth *forward* and *future secrecy* guarantees. It uncovers two new vulnerabilities in Bluetooth's *session establishment*, enabling to reuse of a weak session key across sessions. We show how to exploit these flaws in six attack scenarios to impersonate and MitM arbitrary devices across sessions. Our attacks break Bluetooth's forward and future secrecy as they compromise past and future encrypted messages with novel key reuse attacks. Our findings result from experiments with Bluetooth session establishment on actual devices and inference from the standard. We focused on *SK* as, unlike *PK*, it can be targeted without user interaction, and its entropy can be lowered without violating the standard.

We provide `BLUFFS`, a low-cost and reproducible toolkit to implement, detect, and fix the attacks. The toolkit includes seven original *patches* to manipulate session key derivation and monitor *SK*s by patching the attack device's Bluetooth firmware. It also ships *parsing* and *analysis* scripts to detect the attacks from a pcap file. We use our toolkit to evaluate the BLUFFS attacks on a large

scale. We exploit *eighteen* devices embedding *seventeen* Bluetooth chips from leading hardware and software vendors and estimate the attacks' impact. For example, our threats are effective in all scenarios where at least one of the victims supports LSC and even in scenarios where the victims support SC. These results translate into millions of exploitable devices.

To address the attacks' critical impact, we develop and test a *protocol-level* countermeasure preventing *by-design* the BLUFFS attacks and their root causes. We design an enhanced KDF for LSC employing *fresh*, *mutual*, and *authenticated* session key derivation. We show how to update the LMP protocol and KDF_{LSC} to integrate our fix in a backward compliant way and with minimal overheads. Specifically, we require one extra LMP command, 48 extra bytes sent over the air, 3 specification-compliant function calls, and minimal textual modifications to the standard. We successfully tested our KDF at the protocol level and released it as part of our BLUFFS toolkit. We hope our fix will soon be added to the standard and implemented by the vendors. Moreover, we recommend to vendors implementation-level mitigations that can be adopted while waiting for an update to the standard.

From this work, we learned three key lessons that we want to share: (i) we should pay more attention to session establishment vulnerabilities, attacks, and fixes effective across sessions, (ii) we should agree on the definitions of Bluetooth's forward and future secrecy and update the standard to discuss these definitions and related risks, (iii) we need open-source Bluetooth firmware (Controllers) and better tooling around them to improve the effectiveness, coverage, and speed of our offensive and defensive evaluations.

ACKNOWLEDGMENTS

Work funded by the European Union under grant agreement no. 101070008 (ORSHIN project). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Mingrui Ai, Kaiping Xue, Bo Luo, Lutong Chen, Nenghai Yu, Qibin Sun, and Feng Wu. 2022. Blacktooth: Breaking through the Defense of Bluetooth in Silence. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 55–68.

[2] Daniele Antonioli and Mathias Payer. 2022. On the Insecurity of Vehicles Against Protocol-Level Bluetooth Threats. In *2022 IEEE Security and Privacy Workshops (SPW)*. IEEE, 353–362.

[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. BIAS: Bluetooth impersonation attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 549–562.

[4] 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 Asia conference on computer and communications security (ASIACCS)*.

[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*. 1047–1061.

[6] ARM Developers. 2022. ARM Thumb-2 instruction set. https://developer.arm.com/documentation/ddi0344/k/programmers-model/thumb-2-instruction-set.

[7] Elaine Barker, Lily Chen, Richard Davis, et al. 2018. Recommendation for key-derivation methods in key-establishment schemes. *NIST Special Publication* 800 (2018), 56C.

[8] Christoforus Juan Benvenuto. 2012. Galois field in cryptography. *University of Washington* 1, 1 (2012), 1–11.

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

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

[12] Bluetooth SIG. 2021. Bluetooth Market Update 2021. https://www.bluetooth.com/bluetooth-resources/2021-bmu/.

[13] Bluetooth SIG. 2022. Bluetooth Market Update 2022. https://www.bluetooth.com/2022-market-update/.

[14] Bluetooth SIG. 2022. Reporting Security Vulnerabilities. https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/reporting-security/.

[15] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A formal security analysis of the signal messaging protocol. *Journal of Cryptology* 33 (2020), 1914–1983.

[16] Marco Cominelli, Francesco Gringoli, Paul Patras, Margus Lind, and Guevara Noubir. 2020. Even black cats cannot stay hidden in the dark: Full-band de-anonymization of Bluetooth Classic devices. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 534–548.

[17] Daniele Antonioli (francozappa). 2020. KNOB attack repository on Github. https://github.com/francozappa/knob.

[18] Daniele Antonioli (francozappa). 2021. BIAS attack repository on Github. https://github.com/francozappa/bias.

[19] Dennis Mantz (demantz). 2022. BTBB Wireshark plugin from the Ubertooth libbtbb project. https://github.com/demantz/lmp_wireshark_dissector.

[20] Dor Green. 2022. Pyshark Python packet parser using wireshark's tshark. https://kiminewt.github.io/pyshark/.

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

[22] Electronic Frontier Foundation (EFF). 1998. Cracking DES. https://archive.org/details/crackingdes00elec.

[23] ERNW. 2020. CVE-2020-0022 an Android 8.0-9.0 Bluetooth Zero-Click RCE – BlueFrag. https://insinuator.net/2020/04/cve-2020-0022-an-android-8-0-9-0-bluetooth-zero-click-rce-bluefrag/.

[24] Matheus E Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 2022. BrakTooth: Causing Havoc on Bluetooth Link Manager via Directed Fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*. 1025–1042.

[25] 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.

[26] Simon Hay and Robert Harle. 2009. Bluetooth tracking without discoverability. In *International Symposium on Location-and Context-Awareness*. Springer, 120–137.

[27] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. 2014. BlueID: A practical system for Bluetooth device identification. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2849–2857.

[28] Lin-Shung Huang, Shrikant Adhikarla, Dan Boneh, and Collin Jackson. 2014. An experimental study of TLS forward secrecy deployments. *IEEE Internet Computing* 18, 6 (2014), 43–51.

[29] 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.

[30] Infineon. 2022. CYW20819. https://www.infineon.com/cms/en/product/wireless-connectivity/airoc-bluetooth-le-bluetooth-multiprotocol/airoc-bluetooth-le-bluetooth/cyw20819/.

[31] Internet Engineering Task Force (IETF). 2018. The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446.

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

[33] Jiska YouTube Channel. 2021. InternalBlue Tutorial - 2021 Edition. https://www.youtube.com/watch?v=UANnKx91vyg.

[34] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. 2023. BLEDiff: Scalable and Property-Agnostic Noncompliance Checking for BLE Implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1082–1100.

[35] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. 2006. Breaking ciphers with COPACOBANA–a cost-optimized parallel code breaker. In *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*. Springer, 101–118.

[36] 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.

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

[38] Angela M Lonzetta, Peter Cope, Joseph Campbell, Bassam J Mohd, and Thaier Hayajneh. 2018. Security vulnerabilities in Bluetooth technology as used in IoT. *Journal of Sensor and Actuator Networks* 7, 3 (2018), 28.

[39] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. 2019. InternalBlue Bluetooth binary patching and experimentation framework. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. 79–90.

[40] James L Massey, Gurgen H Khachatrian, and Melsik K Kuregian. 1998. Nomination of SAFER+ as candidate algorithm for the Advanced Encryption Standard (AES). *NIST AES Proposal* (1998).

[41] 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.

[42] Andy Nguyen. 2020. BleedingTooth: Linux Bluetooth Zero-Click Remote Code Execution. https://google.github.io/security-research/pocs/linux/bleedingtooth/writeup.

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

[44] Haram Park, Carlos Kayembe Nkuba, Seunghoon Woo, and Heejo Lee. 2022. L2Fuzz: Discovering Bluetooth L2CAP Vulnerabilities Using Stateful Fuzz Testing. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 343–354.

[45] Trevor Perrin and Moxie Marlinspike. 2016. The double ratchet algorithm. *GitHub wiki* (2016).

[46] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. 2020. Frankenstein: Advanced wireless fuzzing to exploit new Bluetooth escalation targets. In *29th USENIX Security Symposium (USENIX Security 20)*. 19–36.

[47] seemoo-lab. 2021. InternalBlue repository on Github. https://github.com/seemoo-lab/internalblue.

[48] Seemoo-lab. 2022. BTBB plugin for Wireshark 3.6. https://github.com/seemoo-lab/h4bcm_wireshark_dissector.

[49] Ben Seri and Gregory Vishnepolsky. 2017. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/.

[50] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. 2019. BLEEDINGBIT: The hidden Attack Surface within BLE chips. https://armis.com/bleedingbit/.

[51] 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.

[52] Pallavi Sivakumaran and Jorge Blasco. 2021. argXtract: Deriving IoT Security Configurations via Automated Static Analysis of Stripped ARM Cortex-M Binaries. In *Annual Computer Security Applications Conference*. 861–876.

[53] 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.

[54] Jörn Tillmanns, Jiska Classen, Felix Rohrbach, and Matthias Hollick. 2020. Firmware insider: Bluetooth randomness is mostly random. *arXiv preprint arXiv:2006.16921* (2020).

[55] US NSA Research Directorate. 2022. ghidra: a software reverse engineering (sre) suite of tools. https://ghidra-sre.org/.

[56] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. 2021. Method confusion attack on Bluetooth pairing. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1332–1347.

[57] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. 2020. Firmxray: Detecting Bluetooth link layer vulnerabilities from bare-metal firmware. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 167–180.

[58] Wikipedia Community. 2022. Apple silicon. https://en.wikipedia.org/wiki/Apple_silicon.

[59] Wireshark developers. 2022. Wireshark homepage. https://www.wireshark.org/.

[60] Ford-Long Wong and Frank Stajano. 2005. Location privacy in Bluetooth. In *Proceedings of the European Workshop on Security in Ad-hoc and Sensor Networks*. Springer, 176–188.

[61] Yue Zhang and Zhiqiang Lin. 2022. When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3181–3194.

APPENDIX

Here we present the Listings referenced in the paper.

Listing 1: Patch to refuse Peripheral's role switch requests.

```
@ Jumped from 0xA643C (handleLmpSwitchReq)
@ Load second parameter for isMssInstantPassed
ldr r1, [r6, #0x0]
@ Call isMssInstantPassed
bl #0XA63FE
@ Set return value to True
mov r0, #0x1
@ Jump to ROM at 0xA643C+7 in Thumb-2 mode
b #0xA6443
```

Listing 2: Parser's `LmpBase` Class

```
class LmpBase(object):
  """Base Class for LMP Parsing"""
  def __init__(self, pkt):
    self.number = int(pkt.number)
    _tinit = int(pkt.h4bcm.btbrlmp_tid)
    self.tinit = LMP_TRANS_INIT[_tinit]
    self.op = int(pkt.h4bcm.btbrlmp_op)
    if self.op == 127:
      _op_ext = int(pkt.h4bcm.btbrlmp_eop)
      self.op_ext = _op_ext
      self.op_str = LMP_OP_EXT[self.op_ext]
    else:
      self.op_str = LMP_OP[self.op]
```

Listing 3: Parser's `LmpAuRand` Class

```
class LmpAuRand(LmpBase):
  """Parse LMP_au_rand"""
  def __init__(self, packet):
    super().__init__(packet)
    self.aurand = packet.h4bcm.btbrlmp_rand
    self.aurand_ba = bytearray.fromhex(
        self.aurand.replace(":", ""))
```

Listing 4: Excerpt of Kdf's `kdf` function

```
def kdf(LK, AU_RAND, EN_RAND, BTADDR, ENTROPY):
    """Generate KcPrime"""
    BTADDR.reverse()
    _, COF = e1(LK, AU_RAND, BTADDR)
    log.debug("COF: {}".format(repr(COF)))
    # NOTE: redo reverse as it is passed by reference
    BTADDR.reverse()
    Kc = e3(LK, EN_RAND, COF)
    log.debug("Kc: {}".format(repr(Kc)))
    Kc.reverse()
    KcPrime = Kc_to_Kc_prime(Kc, ENTROPY)
    KcPrime.reverse()
    return KcPrime
```

Listing 5: Analyzer's `gen_analysis` function

```
def gen_analysis(PCAP, LK, EXP_SK, BTADD_P):
    """Generate list of sessions and reports"""
    sessions = gen_sessions(PCAP)
    reports = []
    for session in sessions:
        report = gen_report(session, LK, BTADD_P)
        reports.append(report)
    i = 1
    for report in reports:
        print(f"## Begin session: {i}")
        if "keysize" in report:
            print(f"keys: {report['keysize']}")
        if "enrand" in report:
            print(f"enr: {report['enrand'].hex()}")
        if "aurand" in report:
            print(f"aur: {report['aurand'].hex()}")
        if "sk" in report:
            print(f"sk ses: {report['sk'].hex()}")
            # NOTE: check constant SK
            if report["aurand"] == BA_16_ZEROS
              and report["enrand"] == BA_16_ZEROS:
                print(f"sk exp: {EXP_SK.hex()}")
                assert report["sk"] == EXP_SK
        print(f"## End session: {i}\n")
        i += 1
```