Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy

DANIELE ANTONIOLI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
NILS OLE TIPPENHAUER, CISPA Helmholtz Center for Information Security, Germany
KASPER RASMUSSEN, University of Oxford, United Kingdom

Bluetooth (BR/EDR) and Bluetooth Low Energy (BLE) are pervasive wireless technologies specified in the Bluetooth standard. The standard includes key negotiation protocols used to generate long-term keys (during pairing) and session keys (during secure connection establishment). In this work, we demonstrate that the key negotiation protocols of Bluetooth and BLE are vulnerable to standard-compliant entropy downgrade attacks. In particular, we show how an attacker can downgrade the entropy of any Bluetooth session key to 1 byte, and of any BLE long-term key and session key to 7 bytes. Such low entropy values enable the attacker to brute-force Bluetooth long-term keys and BLE long-term and session keys, and to break all the security guarantees promised by Bluetooth and BLE. As a result of our attacks, an attacker can decrypt all the ciphertext and inject valid ciphertext in any Bluetooth and BLE network.

Our key negotiation downgrade attacks are conducted remotely, do not require access to the victims' devices, and are stealthy to the victims. As the attacks are standard-compliant, they are effective regardless of the usage of the strongest Bluetooth and BLE security modes (including Secure Connections), the Bluetooth version, and the implementation details of the devices used by the victims. We successfully attack 38 Bluetooth devices (32 unique Bluetooth chips) and 19 BLE devices from different vendors, using all the major versions of the Bluetooth standard. Finally, we present effective legacy compliant and non-legacy compliant countermeasures to mitigate our key negotiation downgrade attacks.

CCS Concepts: • **Security and privacy → Network security**; **Cryptography**; *Systems security*;

Additional Key Words and Phrases: Security, bluetooth, BLE, key negotiation, downgrade attack, KNOB attack

ACM Reference format:
Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Trans. Priv. Secur.* 23, 3, Article 14 (June 2020), 28 pages.
https://doi.org/10.1145/3394497

1 INTRODUCTION

Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and Bluetooth Low Energy (referred to in the rest of the article as Bluetooth and BLE) are pervasive wireless technologies for personal area networks (PAN). Such technologies are used by billions of devices, including smartphones, laptops,

Authors' addresses: D. Antonioli, École Polytechnique Fédérale de Lausanne (EPFL), Route Cantonale, 1015, Lausanne, Switzerland; email: daniele.antonioli@epfl.ch; N. O. Tippenhauer, CISPA Helmholtz Center for Information Security, Stuhlsatzenhausweg 5, 66123, Saarbrücken, Germany; email: tippenhauer@cispa.saarland; K. Rasmussen, University of Oxford, Parks Road, OX1 3QD, Oxford, United Kingdom; email: kasper.rasmussen@cs.ox.ac.uk.

2471-2566/2020/06-ART14 $15.00
https://doi.org/10.1145/3394497

tablets, wearables, industrial devices, medical devices, and cars [13]. Bluetooth and BLE are fundamentally different technologies using different physical layers, link layers, application layers, and security architectures (see Ref. [42] for a detailed comparison). Ultra low-power wireless devices (such as wearables and IoT sensors) include only BLE, less power-constrained devices (such as laptops, smartphones, and tablets) include Bluetooth and BLE, and devices used to stream content (such as cars and wireless speakers) use only Bluetooth.

Bluetooth and BLE are both specified in the Bluetooth standard [12], maintained and updated by the Bluetooth Special Interest Group (SIG). The standard defines security mechanisms to protect Bluetooth and BLE communication at the link layer, and the correctness of such mechanisms is fundamental to prevent standard-compliant attacks. For example, if a key negotiation protocol in the standard is flawed by design, then all standard-compliant devices are vulnerable to that flaw. In this work, we discuss *downgrade vulnerabilities*, which are present whenever a design enables to insecurely negotiate a weak security mechanism from a list of supported mechanisms. For example, in a cipher downgrade attack, the attacker forces the victims to negotiate Data Encryption Standard cipher (DES) (a weak cipher) instead of Advanced Encryption Standard cipher (AES) (a strong cipher). Backward compatibility is one of the common root causes of downgrade attacks. Several downgrade attacks were found on Secure Sockets Layer (SSL) and Transport Layer Security (TLS), targeting cipher suites, protocol versions, key exchange algorithms, hash functions, and forward secrecy [1]. The Wi-Fi Protected Access 3 (WPA3) protocol, used for Wi-Fi security, was found vulnerable to a version downgrade attack [52] that forces the victim to use a vulnerable version of WPA2 [50, 51]. 5G (LTE) connections can also be downgraded to 2G and 3G networks [45], with known attacks on 2G [25].

The Bluetooth standard specifies link-layer security mechanisms to guarantee authentication, confidentiality, and integrity for Bluetooth [12, p. 1646] and BLE [12, p. 2289]. In particular, the standard defines Bluetooth and BLE key negotiation mechanism to set, among others, the entropy (strength) of the negotiated key. Bluetooth key negotiation is performed during the establishment of a new secure connection to negotiate a variable entropy session key. The session key is first derived from the long-term key (LTK, established while pairing), then its entropy is adjusted according to the negotiated entropy value. BLE key negotiation is part of the BLE pairing process that is used to establish a variable entropy LTK. Whenever two BLE devices establish a new secure connection, a session key is derived from the LTK, and the session key inherits the entropy of the LTK.

In this article, we demonstrate that *the key negotiation protocols of Bluetooth and BLE are insecure by design, and they are vulnerable to standard-compliant entropy downgrade attacks*. In particular, an attacker can downgrade the entropy of any Bluetooth session key to 1 byte without having to know the LTK, and of any BLE long-term and session keys to 7 bytes. Then, the attacker can brute-force the downgraded low-entropy keys, and break the link-layer security guarantees provided by Bluetooth and BLE.

Our downgrade attacks are enabled by three main issues that we identify in the Bluetooth standard: (i) the standard does not mandate to encrypt and integrity protect Bluetooth and BLE key negotiation protocols, (ii) the standard allows to negotiate Bluetooth session keys with 1 byte of entropy and BLE long-term and session keys with 7 bytes of entropy, (iii) the standard does not require to notify the users about the amount of negotiated entropy. The first issue allows a remote attacker to man-in-the-middle (MitM) the key negotiation protocol and to manipulate, among others, the negotiated entropy. The second issue allows the attacker to reduce the entropy to a value that can be brute-forced, as 1 and 7 bytes of entropy are not recommended values for cryptographic keys [8]. The third issue allows the attacker to be stealthy.

As our attacks are *standard-compliant*, they are effective regardless of the Bluetooth and BLE version, the devices' software and hardware manufacturers, and the usage of the strongest Blue-

tooth and BLE security modes such as Secure Connections (SC). Continuing our prior work on Bluetooth [3], in this article, we show that BLE is also vulnerable to entropy downgrade attacks, we extend our Bluetooth contribution by implementing our attacks for Bluetooth SC, by testing our Bluetooth attacks against additional chips and devices, and by experimentally verifying whether (effective) countermeasures were deployed to end-users' devices as a result of our prior work.

We implement our downgrade attacks on Bluetooth and BLE using a combination of techniques and tools dealing with the host and the controller components. The Bluetooth implementation (see Section 5) leverages our implementation of proprietary Bluetooth's security procedures, the InternalBlue toolkit [37], and the implementation of Advanced RISC Machine (ARM) patches for a Nexus 5 and a Bluetooth development board. The BLE implementation (see Section 7) leverages our implementation of modified Linux machines (Linux kernel and bluez), and a custom user-space BLE stack (based on scapy). We use our implementation to evaluate the effectiveness of our attacks on real devices. We successfully attack 38 Bluetooth devices (32 Bluetooth chips) and 19 BLE devices. Our results prove that our attacks are practical and standard-compliant, as we successfully attack Bluetooth and BLE devices from different hardware and software vendors (e.g., Apple, Intel, Microsoft, Qualcomm, Broadcom, Google, Lenovo, and Samsung), implementing different versions of the Bluetooth standard (e.g., 4.0, 4.1, 4.2, 5.0) and using different security modes (e.g., SC).

We summarize our main contributions as follows:

- We identify that the Bluetooth and BLE key negotiation protocols specified in the Bluetooth standard are vulnerable to entropy downgrade attacks. We design attacks capable of lowering the entropy of any Bluetooth session key to 1 byte, and any BLE LTK and session key to 7 bytes. As such low-entropy keys can be brute-forced with reasonable effort, our attacks break all the security guarantees provided by Bluetooth and BLE. Moreover, our downgrade attacks are standard-compliant, transparent to the victims, oblivious to security mode in use, and can be conducted remotely.
- We demonstrate the practical feasibility of our attacks by implementing them for Bluetooth and BLE. Our implementations involve MitM attackers capable of manipulating the key negotiation of Bluetooth (while establishing a secure session) and BLE (while pairing), brute-forcing the keys, and decrypting the traffic exchanged by the victims.
- We show that our attacks pose significant risks to all Bluetooth and BLE users by successfully attacking 38 Bluetooth devices (32 unique Bluetooth chips) and 19 BLE devices. To mitigate such risks, we propose effective standard-compliant and non standard-compliant countermeasures to our attacks, and we test the efficacy of the countermeasures that were rolled after our prior work [3].

The rest of the article is organized as follows. In Section 2, we introduce the relevant background about Bluetooth and BLE. We introduce and compare our key negotiation downgrade attacks in Section 3. We explain the design of the attack for Bluetooth in Section 4, and we present its implementation and evaluation in Section 5. We explain the design of the attack for BLE in Section 6, and we present its implementation and evaluation in Section 7. The attacks' root causes and countermeasures are discussed in Section 8. We present the related work in Section 9. We conclude the article in Section 10.

2 BACKGROUND

In this section, we introduce Bluetooth, BLE, and the Host Controller Interface (HCI).

2.1 Bluetooth BR/EDR

Bluetooth (officially called Bluetooth BR/EDR) is a wireless technology for low-power personal area network. Bluetooth's physical layer uses the 2.4 GHz industrial, scientific and medical (ISM)

band and frequency hopping spread spectrum. A Bluetooth network is called a piconet and uses a master-slave medium access protocol. Each piconet has one master who coordinates up to seven slaves by enforcing a Bluetooth clock signal (CLK). Each Bluetooth device is addressed with 6 byte Bluetooth address (BTADD). The first two bytes (from left to right) of BTADD are defined as non-significant address part (NAP), the third byte as upper address part (UAP), and the last three bytes as lower address part (LAP).

The Bluetooth standard includes link-layer security mechanisms for authentication, confidentiality, and integrity. One of such mechanisms is pairing, which is used by two Bluetooth devices to establish an LTK (known in the standard as link key). There are four types of link keys: initialization unit, combination, and master. In this article, we focus on combination keys, as they are the most secure and widely used. Combination keys are generated as part of Bluetooth's Secure Simple Pairing (SSP). SSP uses Elliptic Curve Diffie Hellman (ECDH) and a key derivation function to compute the link key, and a challenge-response protocol to mutually authenticate the link key.

The Bluetooth standard defines custom security procedures for key agreement, key authentication, and (authenticated) encryption. The standard defines the E_1 procedure to authenticate the link key (K_L), and such procedure works as follows. The master asks to authenticate that a slave owns the link key by sending him a challenge AU_RAND. The slave uses E_1 to compute a Signed Response (SRES) and an Authenticated Ciphering Offset (ACO) from AU_RAND and its Bluetooth address (BTADD_S). The slave sends SRES back to the master, which, in turn, can verify that the slave posses K_L. The standard defines the E_3 procedure to generate session keys from K_L, and such procedure works as follows. E_3 takes as input a Ciphering Offset Number (COF), K_L, and EN_RAND (a public nonce) and produces a session key as output. When K_L is a combination key, COF equals the ACO value computed by E_1. E_1 and E_3 internally use a custom hash function that the standard defines as H. H is based on SAFER+, a block cipher that was submitted as an AES candidate in 1998 [38].

Bluetooth supports legacy and Secure Connections security modes, and the mode is selected based on the devices' capabilities. If the host and the controller of the connecting devices support Secure Connections, then SSP is performed on the P-256 curve, and encryption uses the AES cipher in Counter with cipher block chaining message authentication code (CCM) mode. AES CCM combines AES in counter mode (CTR) for encryption with AES CBC-MAC for authentication, it uses a block size of 128 bit (see Request for Comments (RFC) 3610 [27]). The standard specifies to use AES CCM with MAC of 4 bytes and a length field of 2 bytes. If SC is not supported, then SSP is performed on the P-192 curve, and encryption uses the E_0 stream cipher. E_0 derives from the Massey-Rueppel algorithm [12, p. 1662], and requires synchronization between the master and the slave though their Bluetooth CLK.

2.2 Bluetooth Low Energy (BLE)

BLEis a technology for ultra low power wireless PAN. It was introduced in 2010 with Bluetooth version 4.0 as a simpler and less power consuming alternative to Bluetooth BR/EDR [12]. BLE is supported by millions of high-end and low-end devices including laptops, tablets, mice, keyboards, fitness bands, smartwatches, smart locks, thermostats, and smart TVs. Recently, BLE has been employed in novel ways such as hardware security keys [47], vehicular networks [35], secret handshakes [39], payment systems [21], and biomedical applications [55]. BLE is not compatible with Bluetooth as it uses a different physical layer, link layer, application layer, and security architecture. At the physical layer, BLE uses the 2.4 GHz ISM band with a frequency hopping spread spectrum. At the link layer, it uses a master-slave medium access protocol.

The Bluetooth standard specifies dedicated BLE security mechanisms for confidentiality, authentication, and integrity. Those mechanisms are implemented in the Security Manager (SM)

component [12, p. 2289]. One of such mechanisms is pairing that is used by two BLE devices to establish and authenticate several keys, including a long term key (knowns in the standard as LTK). There are two BLE pairing mechanisms: legacy pairing and SC pairing. Legacy pairing uses a custom key establishment scheme where the LTK is generated from *Short Term Key (STK)* and *Temporary Key (TK)* short-term secrets. Legacy pairing, unless performed with secure out of band (OOB) data, is not secure against eavesdropping and MitM attacks, and should not be used when two devices need a secure connection [43]. Secure Connections pairing was introduced in 2014 with Bluetooth v4.2 to overcome legacy pairing security issues. Secure Connections pairing is considered secure against eavesdropping and MitM attacks, as it generates the LTK using ECDH on the NIST P-256 curve, and it provides mutual authentication of LTK. SC pairing can only be used when both devices support it. Regardless of the pairing mechanism, once two BLE devices share an LTK, they use it multiple times to generate fresh session keys upon secure connection establishment. In particular, the session key is used for AES CCM-based authenticated encryption.

The Bluetooth standard defines two BLE security modes. Mode 1 supports authenticated encryption (AES CCM), while Mode 2 supports only data integrity AES CBC-MAC). Each mode has several levels, and the higher the level the more secure is the connection. An application can enforce a mode and a level to provide specific security guarantees [12, p. 2067]. BLE's strongest security mode is Mode 1 Level 4; it uses authenticated ECDH to establish an LTK with 16 bytes of entropy, and AES CCM to mac-then-encrypt with keys having 16 bytes of entropy [12, p. 2767]. What follows is the complete list of BLE security modes and levels:

Mode 1: Authenticated encryption (AES CCM)
- Level 1: No encryption and no authentication
- Level 2: Unauthenticated pairing with encryption
- Level 3: Authenticated pairing with encryption
- Level 4: Authenticated LE SC, and 128-bit (16-bytes) strength encryption key.

Mode 2: Data integrity only (AES CMAC)
- Level 1: Unauthenticated pairing with data signing
- Level 2: Authenticated pairing with data signing

2.3 The Host Controller Interface (HCI)

Modern implementations of Bluetooth and BLE provide the Host Controller Interface (HCI). This interface allows to separate the Bluetooth and BLE stacks into host and controller components, and reuse components from different manufacturers. Each component has specific responsibilities, i.e., the controller manages low-level radio and baseband operations, and the host manages security procedures and application layer profiles. The host is implemented by user and kernel space programs in the OS of the main CPU, and the controller is implemented in the firmware of the Bluetooth chip. For example, BlueZ and the Linux kernel implement the host component on Linux, and the firmware of an Intel wireless system-on-chip (SoC) implements the controller component. The standard specifies different security architectures for Bluetooth and BLE. All Bluetooth security mechanisms (including key negotiation) are implemented in the controller, while for BLE, some mechanisms (including key negotiation) are implemented in the host and others in the controller.

The host and the controller communicate using the HCI protocol. The HCI protocol is based on commands and events, i.e., the host sends commands to the controller, and the controller uses events to notify the host. HCI can be extended with vendor-specific commands and events. For example, some vendors include special HCI commands to send patches from the host to the controller to update the firmware of the Bluetooth chip without having to change the chip read-only

memory (ROM). The HCI protocol can use different physical transports such as universal asynchronous receiver-transmitter (UART), serial peripheral interface (SPI), and universal serial bus (USB), and can be sniffed using open-source programs such as Wireshark.

3 KEY NEGOTIATION DOWNGRADE ATTACKS ON BLUETOOTH AND BLE

In this section, we introduce the system model, the attacker model, a high-level description of the key negotiation downgrade attacks on Bluetooth and BLE, and a comparison between the two attacks and the relevant prior work.

3.1 System and Attacker Model

System model. Our system model is composed of two victims, Alice and Bob, who want to establish secure wireless connections using Bluetooth or BLE. Without loss of generality, we assume that Alice is the master, and Bob is the slave, and we indicate their Bluetooth addresses with BTADD_M and BTADD_S. Alice and Bob use the strongest security mechanisms provided by Bluetooth and BLE such as Secure Simple Pairing with Passkey entry (i.e., authenticated ECDH) and SC (AES CCM authenticated encryption). Such strong mechanisms should protect Alice and Bob against eavesdropping and MitM attacks. Alice and Bob synchronize themselves using the Bluetooth clock (CLK), and the clock does not provide any security guarantee.

Attacker model. We assume that Charlie, the attacker, wants to target the SC established by Alice and Bob either over Bluetooth or BLE. Charlie wants to decrypt the ciphertext exchanged by the victims to access their secret data and to introduce valid ciphertext in the channel to trick the victims. We assume that Charlie is in range with the victim, and he is capable of eavesdropping (encrypted) packets, manipulating non-encrypted packets, jamming the channel, and crafting valid unencrypted Bluetooth packets. Charlie does not have access to Alice and Bob's devices; he does not know their LTKs and session keys.

3.2 Key Negotiation Downgrade Attack on Bluetooth

The high-level steps to conduct our key negotiation downgrade attack on Bluetooth are presented in the left column of Figure 1. Alice and Bob securely pair in absence of Charlie and establish a strong long-term key (Pairing in Figure 1). Alice and Bob initiate a secure connection to negotiate a session key (Session in Figure 1). As the standard does not mandate to encrypt and integrity protect Bluetooth secure connection establishment, Charlie manipulates the key negotiation such that Alice and Bob negotiate a session key with 1 byte of entropy (the lowest standard-compliant entropy value). Alice and Bob establish a secure connection with a low-entropy session key and start to exchange encrypted traffic (Comm in Figure 1). Charlie computes the session key (with 1 byte of entropy) by eavesdropping the encrypted packets and using them to brute-force the key. Once Charlie knows the session key, then he breaks all Bluetooth security guarantees. We note that Charlie can also target Alice and Bob when they already established a secure connection. In particular, Charlie can jam the connection between Alice and Bob to force them to disconnect and establish a new session and then perform the downgrade attack on the new session.

Our key negotiation downgrade attack on Bluetooth is enabled by the key negotiation protocol specification provided in the Bluetooth standard. The standard does not mandate to protect entropy negotiation (enabling entropy downgrade), and it allows negotiating session key entropy values as low as 1 byte (enabling session key brute-force). Key negotiation is supported since Bluetooth v1.0 (1998), and was introduced to "cope with international encryption regulations and to facilitate security upgrades" [12, p. 1650]. Surprisingly, the standard does not include entropy downgrade in their threat model, and refers to "key size reduction" while the key's entropy is reduced, and not

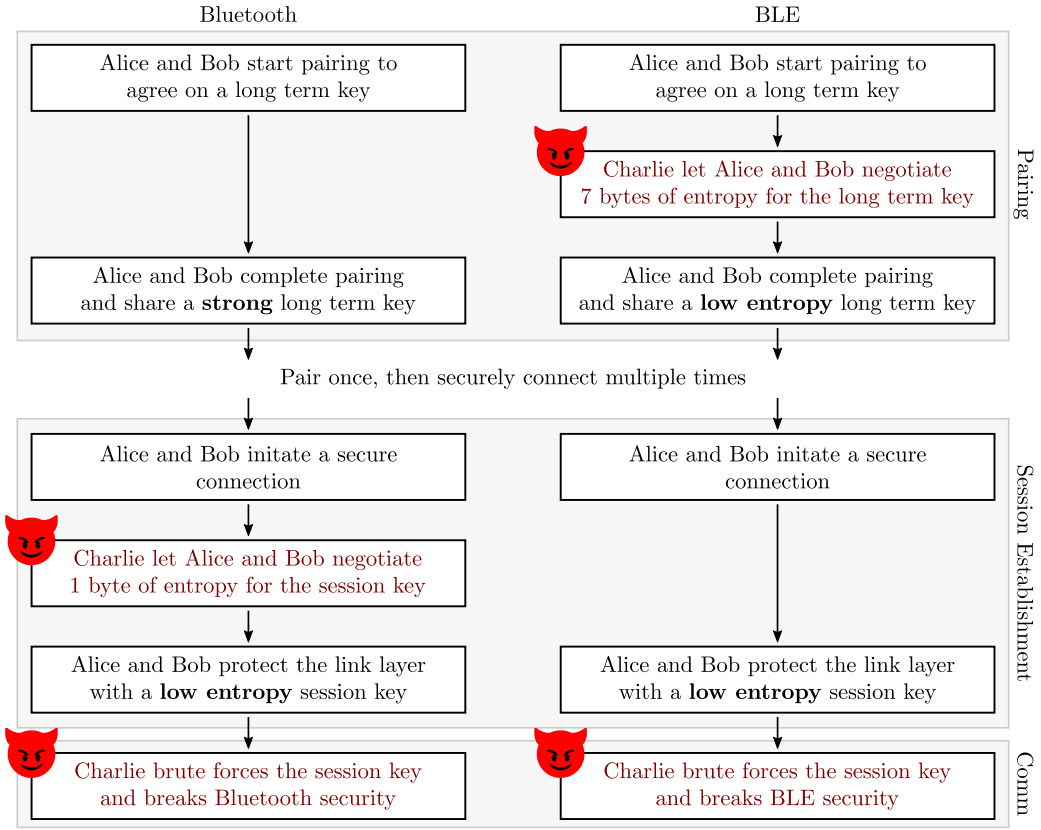

Fig. 1. High-level steps of the key negotiation downgrade attacks on Bluetooth (left column) and BLE (right column). The attacks require different strategies as Bluetooth and BLE are using different security architectures. In the case of Bluetooth, Charlie does not observe Alice and Bob while they are pairing. Once Alice and Bob initiate a new secure connection, Charlie lets them negotiate a session key with 1 byte of entropy. Then, Alice and Bob use the low-entropy session key to protect their communications. Charlie eavesdrops the ciphertext and uses it to brute-force the session key. In the case of BLE, Alice and Bob start the pairing process that includes a LTK negotiation phase, and Charlie lets Alice and Bob negotiate a LTK with 7 bytes of entropy. Then, Alice and Bob establish a secure session using a low-entropy session key, based on the low-entropy LTK. Charlie eavesdrops the ciphertext and uses it to brute-force the low-entropy session term key.

the key size. As the key size always remains 16 bytes, there is no computational overhead when using keys with 16 bytes of entropy compared to keys with lower entropy values.

As the key negotiation downgrade attack is standard-compliant, it affects any Bluetooth device regardless of the implementation details, Bluetooth version number, and security mode. In our experiments (presented in Section 5.4), we successfully attack all Bluetooth security modes, including SC. As key negotiation is transparent to the end-users, our downgrade attacks are unnoticed by the victims. Our attack uncovers non-optimal usage of the Bluetooth LTK established while pairing. Not only is the LTK not used to protect secure connection establishment (including key negotiation), but it also becomes useless after the entropy downgrade attack, because once the entropy of the session key is downgraded it does not matter that the session key was derived from a strong LTK.

3.3 Key Negotiation Downgrade Attack on BLE

The high-level steps to conduct our key negotiation downgrade attack on BLE are presented in the right column in Figure 1. Alice and Bob start pairing to agree upon a LTK (Pairing in Figure 1). BLE key negotiation was introduced with the first release of BLE (Bluetooth v4.0, 2010). BLE key negotiation is performed while pairing and is used to set the entropy of the LTK. As the standard does not require to Integrity protect and encrypt BLE key negotiation, Charlie manipulates it such that Alice and Bob negotiate an LTK with 7 bytes of entropy (the lowest standard-compliant entropy value). Alice and Bob complete pairing and share the low-entropy LTK. Alice and Bob establish a secure connection and they use the low-entropy LTK to derive a low-entropy session key (Session in Figure 1). Alice and Bob start exchanging encrypted packets (Comm in Figure 1) and Charlie eavesdrops the ciphertext and uses it to brute-force the low-entropy session key. Once Charlie knows the session key, he defeats all the security guarantees provided by BLE. We note that Charlie can also target Alice and Bob while they are already paired. In particular, Charlie can force Alice and Bob to re-pair by attempting to pair with Alice as Bob (or vice versa) and failing the LTK authentication procedure.

Our key negotiation downgrade attack on BLE is enabled by a vulnerable key negotiation protocol specified in the Bluetooth standard. The standard does not mandate to protect the feature exchange phase of BLE pairing (enabling entropy downgrade); it allows negotiating LTK entropy values as low as 7 bytes (enabling LTK brute-force). Actually, the standard is not including entropy downgrade in BLE's threat model [12, p. 2309,2311], and as for Bluetooth, it talks in terms of "key size reduction," when in reality, what is reduced is the key's entropy. The size of the key remains 16 bytes, and there is no computational overhead when using keys with 16 bytes of entropy, compared to 7 bytes of entropy.

As our key negotiation downgrade attack on BLE is at the architectural level, any standard-compliant BLE device should be vulnerable, regardless of its implementation details, Bluetooth version, and security mode. In our experiments (presented in Section 7.4), we successfully attack all BLE security modes, including SC. Our attacks are transparent to the end-users, as the standard does not mandate to notify the users about the negotiated entropy while pairing.

3.4 Comparison of Bluetooth and BLE Key Negotiation Downgrade Attacks

Table 1 presents a comparison between Bluetooth key negotiation and BLE key negotiation. The design of Bluetooth key negotiation allows entropy values from 16 to 1 byte and uses an interactive negotiation protocol, visible only from the devices' controllers. By interactive, we mean a negotiation protocol where the participants use successive (downward) entropy proposals to agree upon an entropy value. The key negotiation design for BLE allows entropy values from 16 to 7 bytes and uses a request-response protocol that is visible from the host and the controller. Both key negotiations are neither encrypted nor integrity protected, and they do not require to notify the user about the amount of negotiated entropy. Bluetooth provides the Read Encryption Key Size HCI command to read the session key entropy, and BLE has no HCI entropy query application programming interface (API).

Bluetooth implements key negotiations in the HCI controller (firmware of the Bluetooth chip), using the Link Manager (LM) component and the Link Manager Protocol (LMP), while BLE implements it in the HCI host (main OS of the Bluetooth device), using the Security Manager component and the Security Manager Protocol. The key negotiation downgrade attacks on Bluetooth and BLE are performed during different phases and have different outcomes. The attack on Bluetooth is performed as part of secure connection establishment during session key negotiation. The attacker is capable of reducing the entropy of the session key from 16 bytes to 1 byte and he does not target the

Table 1. Comparison between Bluetooth (Left) and BLE (Right) Key Negotiation

Design	*Bluetooth*	*BLE*
Maximum entropy	16 byte	16 byte
Minimum entropy	1 byte	7 byte
Negotiation logic	Interactive	Request-response
Negotiation visibility	Controller	Host, Controller
Negotiation protection	No	No
Entropy notification	No	No
HCI entropy query	Yes	No
Implementation	*Bluetooth*	*BLE*
HCI	Controller	Host
Component	Link Manager (LM)	Security Manager (SM)
Protocol	Link Manager Protocol (LMP)	Security Manager Protocol (SMP)
Downgrade	*Bluetooth*	*BLE*
Downgrade phase	Connection Establishment	Pairing
Downgrade sub-phase	Session key negotiation	Feature exchange
Downgraded keys	Session key	Long-term and session keys
Downgraded entropy	16 byte → 1 byte	16 byte → 7 byte

As Bluetooth and BLE are different technologies with different security architectures, their key negotiation design, implementation, and downgrade differ significantly.

LTK. On the other hand, the attack on BLE is performed as part of paring during feature exchange. The attacker is capable of reducing the entropy of the LTK from 16 bytes to 7 bytes. Consequently, all the BLE session keys derived from the low-entropy LTK have also 7 bytes of entropy.

4 DESIGN OF OUR DOWNGRADE ATTACKS ON BLUETOOTH

In this section, we describe the design of our key negotiation downgrade attack for Bluetooth. We explain how to negotiate low-entropy session keys and how to brute-force such keys. The implementation and evaluation of our attacks are discussed in Section 5.

4.1 Negotiate Low-Entropy Session Keys for Bluetooth

Alice and Bob (once they completed pairing) negotiate a new session key each time they establish a secure connection. Bluetooth key negotiation is neither encrypted nor integrity protected and allows negotiating session keys with entropy values as low as 1 byte. The same key negotiation is used for legacy and SC modes using the LMP protocol. In the remainder of this section, we indicate a negotiated session key with K'_C and a LTK generated while pairing with K_L.

Figure 2 describes the high-level steps to generate a low-entropy K'_C from K_L. After pairing, Alice and Bob share K_L, and such key has 16 bytes of entropy. Once Alice and Bob start the insecure key negotiation, Charlie manipulates the negotiated entropy to 1 byte (Entropy Negotiation in Figure 2). Then, Alice and Bob compute an intermediate encryption key K_C from K_L and EN_RAND, AU_RAND, and BTADD_S (Encryption Key Generation in Figure 2). Then, they compute K'_C by reducing the entropy of K_C from 16 to 1 byte, and they use K'_C as the session key. As K'_C's length remains 16 bytes, reducing the key's entropy does not provide any performance gain for Alice and Bob. If Alice and Bob support SC, K'_C is used for authenticated encryption with AES CCM, otherwise, for encryption with E_0.

Figure 3 describes the MitM attack used by Charlie to let Alice and Bob negotiate 1 byte of entropy. Without loss of generality, we consider an example where Alice is the master (initiated the

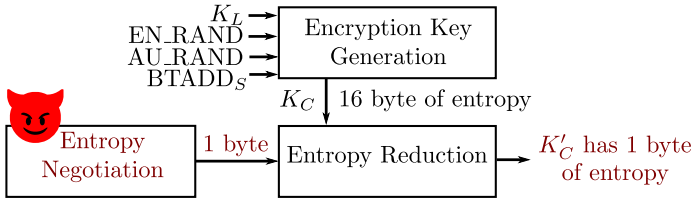

Fig. 2. After pairing, Alice and Bob share K_L (LTK with 16 bytes of entropy). Once they start the insecure key negotiation, Charlie lets them negotiate 1 byte of entropy (Entropy Negotiation). Alice and Bob compute K_C from K_L and other public parameters (Encryption Key Generation), they reduce K_C entropy from 16 bytes to 1 byte (Entropy Reduction), and they use a low-entropy key such as K'_C (session key).

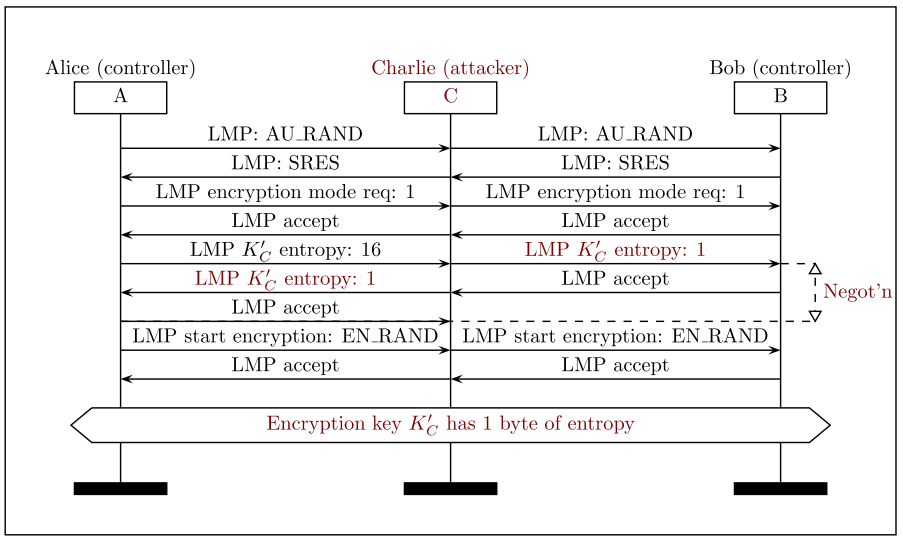

Fig. 3. Adversarial manipulation of the Bluetooth entropy negotiation protocol over LMP. The attacker (Charlie) manipulates the entropy suggested by Alice from 16 to 1 byte. Bob accepts Alice's proposal, and Charlie changes Bob's acceptance to a proposal of 1 byte. Alice accepts the standard-compliant proposal of Bob, and Charlie drops Alice's acceptance message because Bob already accepted Alice's proposal (adversarially modified by Charlie). As a result, Alice and Bob use a K'_C with 1 byte of entropy.

secure connection establishment) and Bob is the slave. Both devices support standard-compliant entropy values between 1 and 16 bytes. The MitM attack works as follows. Charlie relays the first four messages, which allows Alice to authenticate that Bob possesses the correct K_L, and to initiate link-layer encryption. Then, Charlie tampers with the next three messages to modify the negotiated entropy (Negot'n in Figure 3). Alice proposes 16 bytes of entropy (LMP: K'_C entropy: 16), and Charlie changes such entropy value to 1 byte and forwards the message to Bob. Bob sends an acceptance message, and Charlie modifies it into an entropy proposal message with 1 byte of entropy and sends the message to Alice. Alice sends an acceptance message to Bob, and Charlie drops it, as Bob has already accepted the proposal with 1 byte of entropy. As a result, Charlie forces Alice and Bob to negotiate 1 byte of entropy for K'_C.

It is reasonable to think that Alice and Bob could mitigate the key negotiation downgrade by terminating a connection with a low-entropy session key from the application layer. However, the standard states the following: "The possibility of a failure in setting up a secure link is an

unavoidable consequence of letting the application decide whether to accept or reject a suggested key size" [12, p. 1663]. This statement is ambiguous because it is not clear what the definition of "application" is in that sentence. As we show in Section 5.4, this ambiguity results in no-one being responsible for terminating connections with low-entropy keys.

The entity that decides whether to accept or reject the entropy proposal is the firmware of the Bluetooth chip by setting two parameters defined as L_{min} and L_{max} and managing the entropy proposals. The "application" (intended as the Bluetooth application running on the OS using the firmware as a service) cannot check and set L_{min} and L_{max}, and is not directly involved in the entropy acceptance/rejection choice (that is performed by the firmware). The application can interact with the firmware using the HCI protocol. In particular, it can use the HCI Read Encryption Key Size request to check the amount of negotiated entropy *after* the Bluetooth connection is established and theoretically abort the connection. This check is neither required nor recommended by the standard as part of the key negotiation protocol.

4.2 Brute-Forcing the Low-Entropy Bluetooth Session Key

Once Charlie manages to reduce the entropy of K'_C (session key) to 1 byte, he can trivially brute-force it, i.e., brute-force 1 value from 256 candidates. Charlie can do it in parallel, and he does not have to know what type of application-layer traffic is exchanged, because he can use well-known Bluetooth fields as oracles, e.g., logical link control and adaptation protocol (L2CAP) and radio frequency communication (RFCOMM) headers. In the following, we describe how Charlie brute-forces session keys with 1 byte of entropy when Alice and Bob are using SC or legacy security. More details about our brute-force implementation are discussed in Section 5.

Alice and Bob, according to their capabilities, either use legacy security or SC mode. Each mode has a different entropy reduction function and cipher suite. In the case of legacy security, K'_C is computed using Equation (E_s). N is the negotiated entropy (an integer between 1 and 16), and $g_1^{(N)}$ is a polynomial of degree $8N$ used to reduce the entropy of K_C from 16 bytes to N bytes. The result of the reduction is encoded with a block code $g_2^{(N)}$, a polynomial of degree less or equal to $128 - 8N$. The values of those polynomials depend on N, and they are tabulated in Ref. [12, p. 1668].

$$K'_C = g_2^{(N)} \otimes \left(K_C \bmod g_1^{(N)}\right) \tag{E_s}$$

Based on Equation (E_s), when K'_C's entropy is 1 byte, then we compute the 256 candidate keys by multiplying all the possible 1 byte reductions $K_C \bmod g_1^{(1)}$ (the set `0x00`...`0xff`) with $g_2^{(1)}$ (`0x00e275a0abd218d4cf928b9bbf6cb08f`).

In the case of SC, the entropy reduction procedure computes K'_C by setting the $16 - N$ least significant bytes of K_C to zero (`0x00`). Hence, when K'_C's entropy is 1 byte, we brute-force the key against the 256 key candidates in the `0x00`...`0xff` set.

5 IMPLEMENTATION AND EVALUATION OF OUR DOWNGRADE ATTACKS ON BLUETOOTH

In this section, we describe the implementation of our key negotiation downgrade attacks on Bluetooth. We explain our attack scenario, how we implement our attack for legacy and SC. Then, we describe how we use our implementation to successfully attack 38 Bluetooth devices (32 different Bluetooth chips), and we present our evaluation results. Countermeasures to our attacks are discussed in Section 8.2.

Table 2. Devices Used to Implement Our Key Negotiation Downgrade Attacks

		Bluetooth		
Device	OS (Host)	Chip (Controller)	Version	Secure Connections
CYW920819	ThreadX	CYW20819	5.0	Yes
Nexus 5	Android 6.0.1	BCM4339	4.1	No
Thinkpad X1	Linux 4.14	Intel 7265	4.2	Host only

We show the device name, OS (Host), Chip (Controller), Bluetooth version, and Secure Connections support.

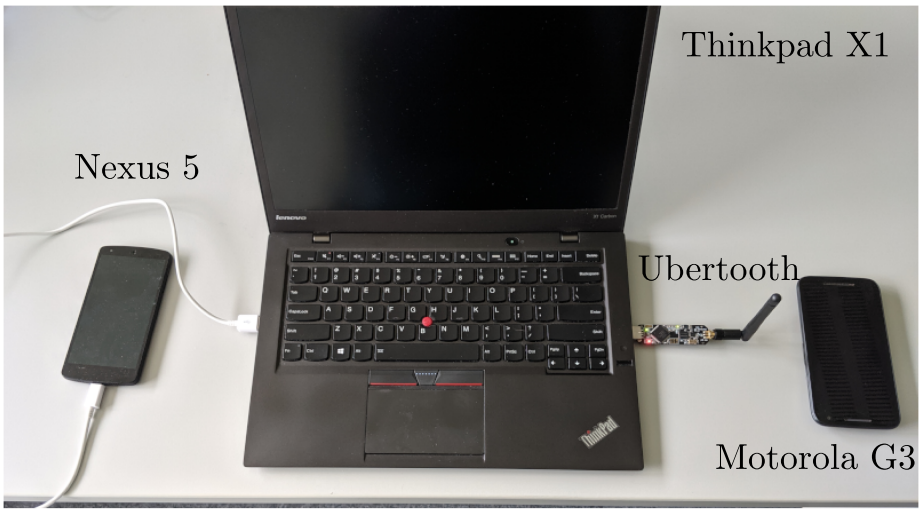

Fig. 4. Our attack scenario for Bluetooth (legacy security) and BLE. In the Bluetooth case, we use a Nexus 5 both as a victim and as the attacker, a Motorola G3 as a victim, and a Thinkpad X1 laptop to patch the Nexus 5's Bluetooth firmware. In the BLE case, we use a Thinkpad X1 both as the attacker and as a victim. The laptop runs our custom Linux kernel and user-space BLE stack. We eavesdrop Bluetooth and BLE packets using an Ubertooth One.

5.1 Bluetooth Attack Scenarios

To describe our implementation, we use an attack scenario for legacy security and another one for SC. Table 2 shows the involved devices with their relevant technical specifications. We now describe the two attack scenarios.

Legacy security. In this attack scenario, Charlie MitMs a secure connection between a Nexus 5 and another victim device. As the Nexus 5 does not support SC, the victims use legacy security mechanisms. In particular, they use SSP on the P-192 curve, the entropy reduction function of Equation (E_s), and E_0 for encryption. In our attacks, we use the Nexus 5 both as a victim and as the attacker. As key negotiation is implemented in the Bluetooth firmware, we use a Thinkpad X1 laptop to patch the Nexus 5 Bluetooth firmware at runtime. We eavesdrop the Bluetooth packets using an Ubertooth One [41]. To the best of our knowledge, Ubertooth One does not capture all Bluetooth packets, but it is the only open-source, low-cost, and practical eavesdropping solution for Bluetooth. Figure 4 shows a picture of an attack where we test if a Motorola G3 is vulnerable to our attack.

Secure Connections. In this attack scenario, Charlie MitMs a secure connection between a CYW920819 development board and a victim device that supports SC. In this case, the victims use SSP on the P-256 curve, the entropy reduction function who zeros least significant bytes, and AES CCM for authenticated encryption. In our attack, the CYW920819 is connected to our Thinkpad laptop, and both devices are used both as a victim and as the attacker. In particular, we use the laptop as a Bluetooth host and to patch the board Bluetooth firmware.

In the remainder of this section, we describe how we implement low-entropy key negotiation and session key brute-force for legacy security and SC. We use our implementations to successfully attack 38 Bluetooth devices, and the results are presented in Section 5.4.

5.2 Implement Low-Entropy Key Negotiation for Bluetooth

As Bluetooth key negotiation is implemented in the Bluetooth firmware (controller), we implement low-entropy key negotiation on legacy security by patching the Nexus 5 Bluetooth firmware, and on SC by patching the CYW920819 board Bluetooth firmware. Our firmware patches are not changing the key negotiation logic but they are introducing extra code in the firmware to modify all the incoming and outcoming negotiation packets (LMP packets). Our implementation allows to reliably simulate over-the-air MitM attacks and quickly test if a legacy or SC device is vulnerable to our attack.

We implement our Bluetooth firmware patches by using InternalBlue [37]. InternalBlue is an open-source toolkit to interact with several Broadcom (Cypress) chips including the ones in our Nexus 5 (BCM4339 with an ARM Cortex M3) and CYW920819 board (CYW20819 with an ARM Cortex M4). InternalBlue provides an API to take advantage of a Broadcom proprietary patching mechanism called patchROM. With patchROM, we can write our own ARM patches, load them into the chip RAM at runtime, and then jump to our patches from arbitrary addresses. This mechanism is very powerful because it allows, among others, to override function calls and redirect the control flow of the firmware. InternalBlue provides an API to write and read the firmware RAM at runtime. We use such API to monitor and modify the relevant firmware data structures, e.g., we can change the minimum and maximum entropy supported by the Nexus 5 and the board.

After reverse-engineering part of the Nexus 5 and CYW920819 firmware with Ghidra [48], we manage to find the addresses of the LMP dispatchers functions. Such functions are called whenever the firmware receives an LMP packet, and before the firmware transmits any LMP packet. By patching the dispatchers, we are able, among others, to manipulate any LMP packet before it is sent and before it is received, without affecting the firmware key negotiation logic. In particular, we patch all the outgoing LMP encryption key size requests such that the proposed entropy is always 1, all the incoming LMP accept encryption key size requests into LMP key size request with 1 byte of entropy, and all outgoing LMP accept encryption key size request into LMP preferred rate (to simulate a drop). As a result, our two victims are tricked to negotiate 1 byte of entropy as in Figure 3.

Our patches cover all the legacy security and SC attack scenarios. There are four attack scenarios per mode: the attacker is the master/slave and initiates the entropy negotiation, and the attacker is the master/slave and does not initiate the entropy negotiation. The main differences between the patches are the LMP transaction ID in use (TID = 0/1 when transaction is initiated by master/slave), and the order of incoming and outgoing entropy negotiation LMP packets.

5.3 Implement Session Key Brute-Force for Bluetooth

In this section, we explain how we brute-force K'_C (session keys) with 1 byte of entropy for legacy and SC, and how we validate such brute-forced values. Our implementation is free and open-source, and is available at https://github.com/francozappa/knob.

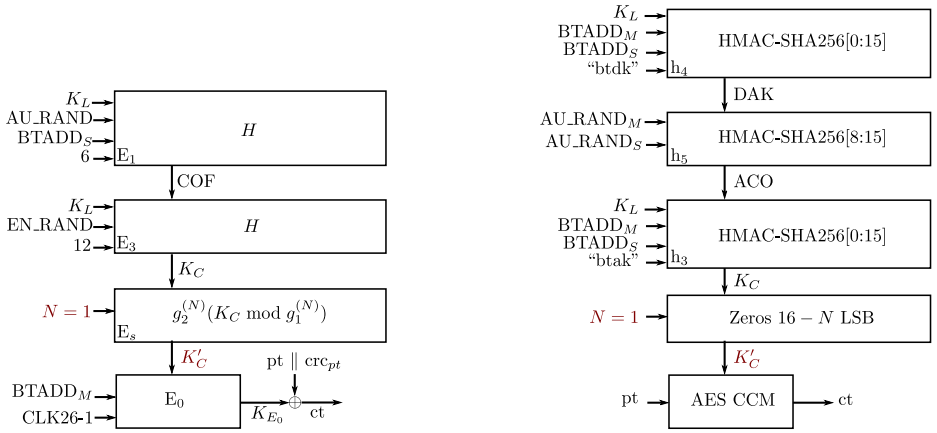

Fig. 5. Session key computation for legacy security (left column) and SC (right column). Legacy security uses custom security procedures (E_1, E_3, E_s) and a stream cipher (E_0) that is not FIPS compliant. SC uses HMAC-SHA256 (square brackets denote hash slicing), a zeroing entropy reduction function, and AES CCM. The AES CCM internal parameters are not showed in the Figure as they are not relevant for the session key's computation. In both cases, the attacker sets 1 byte of entropy ($N = 1$) for the session key (K'_C). COF, DAK, and ACO refer to intermediate value names from the standard.

To brute-force K'_C, we develop a Python script. The script works by testing the decryption of one (or more) ciphertext against K'_C's candidates. There are 256 candidates for legacy security and 256 candidates for SC, and the candidates are precomputed and stored in memory to speed up comparisons. For example, for legacy security, we use our brute-force script to decrypt a file sent from the Nexus 5 to the Motorola G3 using the object exchange (OBEX) profile, after the negotiation of a K'_C with 1 byte of entropy. The file contains the following ASCII text: aaaabbbbccccdddd.

If the victims are using legacy security, then K'_C (1 byte of entropy) is computed from K_L (16 bytes of entropy) and other parameters as in the left column of Figure 5. E_1 computes a COF from K_L, AU_RAND (nonce), and BTADD$_S$. The COF is used together with K_L, and EN_RAND (nonce) by E_3 to compute K_C (16 bytes of entropy). E_s reduces K_C entropy to 1 byte, using modular arithmetic over polynomials in Galois fields, and outputs K'_C. K'_C is used together with BTADD$_M$ and CLK by E_0 to produce the stream key K_{E_0}. E_1 and E_3 internally use a custom Bluetooth hash function, denoted in the standard with H:

$$SRES||ACO = H(K_L, \text{AU\_RAND}, \text{BTADD}_S, 6) \quad (E_1)$$

$$K_C = H(K_L, \text{EN\_RAND}, \text{COF}, 12) \quad (E_3)$$

H internally uses SAFER+, a block cipher submitted as an AES candidate in 1998 [38] and SAFER+' (SAFER+ prime). Both ciphers are used with a block size of 128 bit, and 8 rounds in ECB mode. SAFER+' is a modified version of SAFER+ such that the input of the first round is added to the input of the third round, and such modification is introduced to avoid SAFER+' being used for encryption [12, p. 1677]. We note that the standard indicates SAFER+ and SAFER+' with A_r and A_r' [12, p. 1676]. Figure 6 shows how H uses SAFER+, SAFER+', and other blocks to compute COF (E_1, left side) and K_C (E_3, right side). The E block is an extension block that transforms the 12-byte COF or the 6-byte BTADD$_S$ into a 16-byte sequence using modular arithmetic. The O block is offsetting K_L using algebraic (modular) operations and the largest primes below 257 for which 10 is a primitive root. The remaining block is performing a sum mod 256.

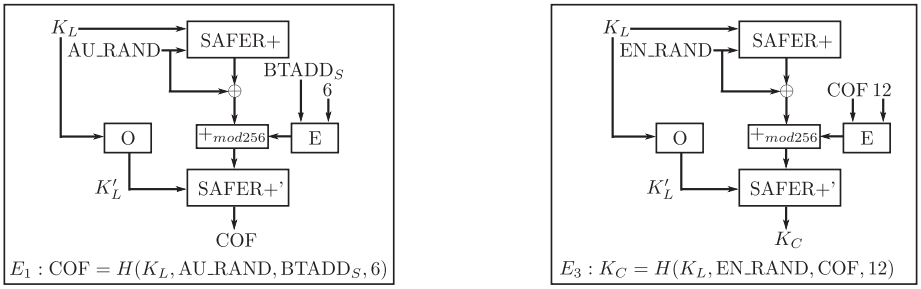

Fig. 6. Bluetooth custom H hash function used by E_1 (left) and E_3 (right) for legacy security. H is based on SAFER+ and SAFER+'.

If the victims are using SC, then K'_C (1 byte of entropy) is computed from K_L (16 bytes of entropy) and other parameters as in the right column of Figure 5. h_4 computes a Device Authentication Key (DAK) from K_L, BTADD$_M$, BTADD$_S$, and the "btdk" string. DAK is used together with AU_RAND$_M$ and AU_RAND$_S$by h_5 to compute ACO. ACO is used together with K_L, BTADD$_M$, BTADD$_S$, and the "btak" string by h_3 to compute K_C (16 bytes of entropy). K'_C (1 byte of entropy) is computed by zeroing the 15 least significant bytes of K_C, and then used as a key for AES CCM authenticated encryption. The h_3, h_4 , and h_5 Bluetooth hash functions internally use HMAC-SHA256, a keyed hash function producing 32 bytes of output. The hash functions use one parameter as HMAC key and concatenate the others to produce the input to be hashed (e.g., h_4 keys HMAC with K_L and uses as input the concatenation of "btdk", BTADD$_M$, and BTADD$_S$). The square brackets in Figure 5 indicate that the hash is sliced (e.g., h_4 outputs DAK by taking the leftmost 16 bytes of the hash).

To validate the brute-forced keys, we implement and test against the standard vectors all the blocks of Figure 5. For legacy security, we implement SAFER+ and SAFER+' including the round computations and the key scheduling algorithm in Python. We use such blocks to implement H, which, in turn, allows implementing E_1 and E_3. We implement E_s using the BitVector [31] Python module to perform modular arithmetic in the Galois field. For the E_0 cipher, we use an open-source implementation [16]. For SC, we implement HMAC-SHA256 and AES CCM using the Python cryptography module [7], and we implement the zeroing entropy reduction function in Python.

5.4 Evaluation of Key Negotiation Downgrade Attacks on Bluetooth

We test if a target victim is vulnerable to our key negotiation downgrade attack as follows:

(1) We pair the Nexus 5 or the CYW920819 board with the victim target device
(2) We patch the Nexus 5 or the CYW920819 firmware before the session key negotiation, and we activate InternalBlue's LMP and HCI monitoring.
(3) We start over the air sniffing with an Ubertooth One (using UAP and LAP flags).
(4) We request a connection from the Nexus 5 or the board to the target victim (or vice versa) to trigger the session key negotiation over LMP.
(5) Our firmware patches change the LMP packets as Charlie does in Figure 3.
(6) If the target victim successfully connects, then it is vulnerable to our downgrade attack, and we can brute-force the session key and decrypt the ciphertext captured with the Ubertooth One.

Table 3 presents the results of our key negotiation downgrade attacks on Bluetooth. We extend our evaluation presented in Ref. [3] by successfully attacking 38 Bluetooth devices from Anker, Apple, Bose, Lenovo, LG, HP, Motorola, Nokia, OnePlus, Philips, Plantronics, Samsung, Sennheiser,

Table 3. Key Negotiation Downgrade Attack Evaluation on Bluetooth

Chip	Device(s)	K'_C Entropy
Bluetooth version 5.0		
Apple A1865	iPhone X	1 byte
Apple 339S00428	MacBookPro 2018	1 byte
Mediatek MT6762	LG K40	3 bytes
Snapdragon 660	Xiaomi MI A2	1 byte
Snapdragon 835	Pixel 2, OnePlus 5	1 byte
Snapdragon 845	Galaxy S9	1 byte
Bluetooth version 4.2		
Apple 339S00045	iPad Pro 2	1 byte
BCM43438	RPi 3B, RPi 3B+	1 byte
BCM43602	iMac MMQA2LL/A	1 byte
CSR 11393	Sennheiser PXC 550	1 byte
CSR 11836	Bose SoundLink revolve	1 byte
CSR 12942	Sony WH-100XM3	1 byte
Exynos 7570	Galaxy J3 2017	1 byte
Intel 7265	Thinkpad X1 3rd, Dell Latitude E7250	1 byte
Intel 8260	HP ProBook 430 G3	1 byte
Intel 8265	Thinkpad X1 6th	1 byte
Snapdragon 625	Xiaomi Mi Max 2	1 byte
Bluetooth version 4.1		
BCM4339 (CYW4339)	Nexus 5, iPhone 6	1 byte
Snapdragon 210	LG K4	1 byte
Snapdragon 410	Motorola G3, Galaxy J5	1 byte
Bluetooth version $\leq$ 4.0		
Apple W1	AirPods	7 bytes
BCM20730	Thinkpad 41U5008	1 byte
BCM4329B1	iPad MC349LL	1 byte
Broadcom 8721	Anker A7721, Thinkpad KT-1255	1 byte
Broadcom 20702	MacBookAir Mid 2012	1 byte
CSR 6530	Plantronics BackBeat 903+	1 byte
CSR 8648	Philips SHB7250+	1 byte
Exynos 3475	Galaxy J3 2016	1 byte
Intel Centrino 6205	Thinkpad X230	1 byte
Snapdragon 200	Lumia 530	1 byte
Snapdragon 615	Galaxy A7	1 byte
Snapdragon 800	LG G2	1 byte

The first column shows the Bluetooth chip, the second column the device(s) using such chip, and the third column K'_C (session key) downgraded entropy. All 32 chips (38 devices) that we test are vulnerable, and they accept to downgrade K'_C's entropy from 16 bytes to values as low as 1 byte. Our downgrade attack is effective regardless of hardware and software manufacturer and the Bluetooth version.

Sony, Raspberry Pi, Xiaomi, and 32 unique Bluetooth chips from Apple, Broadcom (Cypress), Cambridge Silicon Radio (CSR), Exynos, Mediatek, Intel, and Qualcomm. Table 3 is structured as follows. The first column contains the Bluetooth chip, the second column contains the devices that we test using such chip, and the third column contains the minimum entropy value accepted by a

chip for K'_C (session key). The rows are grouped in four blocks, and each block contains devices using the same Bluetooth version, e.g., version 5.0, 4.2, 4.1, and ≤ 4.0.
From Table 3's third column, we show that all the chips that we test are vulnerable to our downgrade attacks. In particular, all of them accept to reduce K'_C entropy to 1 byte, except the Mediatek MT6762 who accepts at least 3 bytes, and the Apple W1 who accepts at least 7 bytes. We note that using three or seven bytes of entropy is not recommended for cryptographic keys [8, 11]. Table 3 also confirms that our downgrade attack is standard-compliant, as it affects any Bluetooth device (regardless of its hardware and software manufacturers), and is effective across the major Bluetooth versions (5.0, 4.2, 4.1, ≤ 4.0). As a consequence of our results, all standard-compliant Bluetooth devices are potentially at risk.

After we disclosed our findings in Ref. [3], several vendors, such as Apple, Google, and Linux provided mitigations for the downgrade attacks on Bluetooth. We discuss such mitigations in Section 8.2.

6 DESIGN OF OUR DOWNGRADE ATTACKS ON BLE

In the last two sections, we describe our key negotiation downgrade attacks on Bluetooth, and now we switch to BLE, which is a different technology than Bluetooth. In particular, BLE has a different security architecture than Bluetooth, and we cannot reuse what we present for Bluetooth to attack BLE. In the following section, we describe the design of our key negotiation downgrade attacks for BLE. In particular, we show how an attacker can downgrade the entropy of any BLE LTK and session key to 7 bytes and how he can brute-force such keys. The implementation and evaluation of our attacks are discussed in Section 7.

6.1 Negotiate Low-Entropy Long-Term Keys for BLE

BLE's LTK negotiation is performed as part of pairing (as described in Section 3.3). Below, we summarize the BLE pairing phases [12, p. 2296]. We note that BLE pairing is always initiated by the master.

- Phase 1: *Feature exchange (including key negotiation).* In this phase the master and the slave exchange input-output capabilities (IO), authentication requirements (AuthReq), LTK's entropy proposals (KeySize), and the initiator and responder key distribution flags (InitKeys, RespKeys). The features are exchanged in clear text and without integrity protection using the SMP protocol. The master sends an SMP Pairing Request and the slave sends an SMP Pairing Response.
- Phase 2: *Key establishment and optional authentication.* In this phase, the master and the slave use either legacy or SC procedures to establish and optionally authenticate the LTK. In the case of legacy pairing, the LTK is derived from a custom BLE key agreement scheme, and is authenticated using one association method within Just Works, Passkey entry, and OOB data. For SC, the LTK is derived using ECDH and a key derivation function, and authenticated using one association method within Just Works, Passkey Entry, Numeric comparison, and OOB data. In both cases, LTK has KeySize bytes of entropy, and the standard-compliant KeySize values are from 7 to 16 bytes.
- Phase 3: *Key distribution over encrypted link.* In this phase, the master and the slave perform the encryption procedure to derive a session key (SK) from LTK and encrypt the link-layer traffic using AES CCM. We note that SK entropy is the same as LTK (set according to KeySize). Then, the master and the slave exchange more keys according to InitKeys and RespKeys flags exchanged in Phase 1.

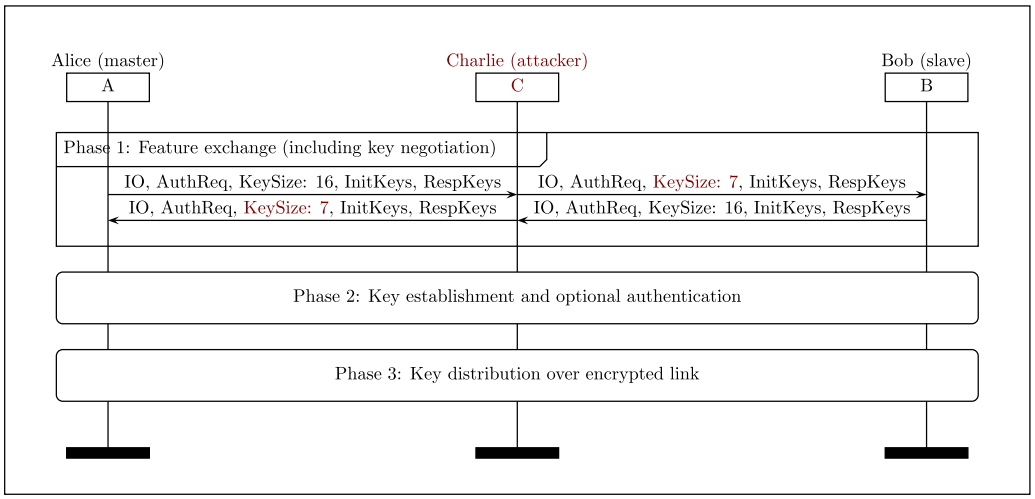

Fig. 7. Charlie let Alice and Bob negotiate an LTK with 7 bytes of entropy. As the feature exchange phase of BLE pairing is not protected, Charlie MitMs Alice and Bob and the entropy values declared by Alice and Bob to the minimum standard-compliant value (KeySize: 7).

The main issue with the Bluetooth standard is that the BLE feature exchange phase is neither encrypted nor integrity protected, and it allows negotiating LTK entropy values between 7 and 16 bytes. Then, the LTK is used to derive SK that inherit the entropy of LTK. Hence, an attacker can manipulate the feature exchange phase while two victims are pairing to let them negotiate an LTK with 7 bytes of entropy and then brute-force the session key.

Figure 7 shows how Charlie performs an MitM attack to reduce LTK's entropy to 7 bytes during the feature exchange phase. Alice (master) sends an SMP Pairing Request message containing IO, AuthReq, KeySize, InitKeys, and RespKeys. Charlie modifies Alice's supported KeySize from 16 to 7 bytes and then sends the modified request to Bob (slave). Then, Bob sends an SMP Pairing Response message containing his capabilities, and Charlie changes Bob's KeySize from 16 to 7 bytes and sends the modified response to Alice. As a result, Alice and Bob agree upon an LTK with 7 bytes of entropy and all the session keys derived from it also have 7 bytes of entropy. The downgrade is unnoticed by Alice and Bob, as the standard does not require to notify the users about LTK entropy.

The Bluetooth standard comments on the security of BLE's feature exchange (Phase 1) and key establishment and optional authentication (Phase 2) stating that "Phase 1 and Phase 2 may be performed on a link that is either encrypted or not encrypted" [12, 2296]. In our BLE experiments, presented in Section 7.4, *none of the devices that we test are encrypting Phase 1 and 2*. We are not surprised by this outcome, as the standard does not require two unpaired BLE devices to share a secret key and to encrypt Phase 1 and Phase 2, two devices need a key. Even if a pre-shared key would be available before pairing, the encryption of Phases 1 and 2 is only "obfuscating" the entropy downgrade problem, because if the cipher in use is malleable, then Charlie can still downgrade LTK's entropy by directly manipulating the ciphertext. What is needed in this case is *data integrity*, and the standard is not recommending to integrity protect Phase 1 and Phase 2.

6.2 Brute-Forcing the Low-Entropy BLE Long-Term Key

Once Charlie can reduce the entropy of LTK to 7 bytes, he can brute-force LTK and the current SK derived from LTK. Once Charlie knows SK, he can decrypt all the ciphertext exchange by Alice

and Bob and introduce valid ciphertext in the session. At the time of writing (2019), and since the mid-nineties, 7 bytes of entropy are not sufficient for a cryptographic key [11]. Prior work has succeeded in brute-forcing keys with 7 bytes of entropy with increasing efficacy, e.g., specialized hardware [34] and cloud infrastructure [54]. Furthermore, considering current best practices [8], any key with less than 14 bytes of entropy is not considered secure for symmetric encryption.

7 IMPLEMENTATION AND EVALUATION OF OUR DOWNGRADE ATTACKS ON BLE

In this section, we describe the implementation of our key negotiation downgrade attack on BLE. We explain our attack scenario, and how we implement our attack for legacy and SC. Then, we describe how we use our implementation to successfully attack 19 BLE devices, and we present our evaluation results.

7.1 BLE Attack Scenario

Our reference attack scenario uses a Thinkpad X1 laptop running Linux both as a victim and as the attacker. We attack legacy and SC, and we act as a BLE central or as a BLE peripheral according to the role of the victim device. Typically, a central device is higher-end than the peripheral one. The goal of our attack is to let the target victim negotiate an LTK with 7 bytes of entropy with our laptop as in Figure 7. If this is the case, all subsequent SC between our laptop and the target victim would use a fresh SK with 7 bytes of entropy. During our experiments, we encountered target victims not supporting security at all, and we consider such devices our scope of this work.

7.2 Manipulation of the BLE LTK

To manipulate LTK's entropy negotiation, the attacker has to manipulate the SMP Pairing Request and Pairing Response exchanged during the BLE pairing feature exchange. As described in Section 6.1, feature exchange is neither encrypted nor integrity protected and allows downgrading LTK's entropy value to 7 bytes. As BLE pairing is implemented in the Host (OS), we have to modify the OS running on our attack device (Linux laptop). In particular, we implement the low-entropy feature exchange in our Linux kernel (`linux-4.14.111`) by modifying the BLE sub-system, and in user-space by developing a custom BLE user-space stack. Both implementations can target legacy and SC, and can reliably simulate MitM key negotiation downgrade attacks on the feature exchange. In the following two paragraphs, we provide more details about our implementations.

Custom Linux kernel. The Linux kernel contains an open-source implementation of the BLE Host. In particular, it includes an implementation of the SMP protocol in `net/bluetooth/smp.c`. We change such implementation, and we are capable of testing various capabilities declarations using dedicated kernel patches. In particular, our BLE key negotiation patch enables to propose arbitrary entropy values for LTK (e.g., KeySize: 7 of Figure 7). This is accomplished by changing line 3424 of `net/bluetooth/smp.c` to `SMP_DEV(hdev)->max_key_size = 7;` and recompiling the kernel.[1]

Custom Linux BLE user-space stack. As Linux offers the possibility to manage BLE from user-space, we develop a custom BLE stack. Our stack builds on top of PyBT [44], an open-source Python package that internally uses scapy [10]. PyBT provides a minimal BLE stack, and we extend it with several APIs. For example, our BLE key negotiation API allows negotiating any LTK entropy value during the feature exchange phase.

Using our Linux laptop both as an attacker and a victim, we can quickly test if a target BLE victim is vulnerable to our entropy downgrade attack using the following steps:

[1]Unmodified code: https://elixir.bootlin.com/linux/v4.14.111/source/net/bluetooth/smp.c#L3424.

(1) On the laptop, we either activate our custom kernel patch or our BLE user-space stack.
(2) We make the target victim discoverable.
(3) If the target victim is a peripheral (slave), we act as a central (master); otherwise, we act as a peripheral.
(4) We complete the feature exchange phase by proposing 7 bytes of entropy (KeySize: 7).
(5) If the victim device is vulnerable it completes pairing otherwise it sends an SMP Pairing Failed message with reason "Encryption Key Size" (SMP opcodes `0x05` and `0x06`) [12, p. 2384].

7.3 Brute-Forcing the Low-Entropy LTK

We implement all the necessary functionality to derive LTK for legacy and SC (ECDH), reduce the entropy of LTK to an arbitrary value, derive SK from the LTK, and generate valid ciphertext and plaintext. In the rest of this section, we provide more details about such implementations.

When the victims are using SC, the LTK is computed from a shared secret generated using ECDH. The standard defines this key derivation function as f5 [12, p. 2303]. f5 uses AES CMAC, a message authenticated code based on AES in CBC mode, to compute the LTK. f5 takes as inputs the ECDH secret, BTADD_M BTADD_S, a master nonce and a slave nonce, and outputs the LTK. On the other hand, when the victims are using legacy security, the LTK is computed using a TK and two nonces (Mrand, Srand). The standard defines this key derivation function as s1, and s1 internally uses AES128. We implement f5 and s1 using the python cryptographic module [7], and we successfully test our implementations against the test vectors in the standard.

The BLE entropy reduction function is straightforward to implement as it requires to zero out the most significant bytes of LTK according to the negotiated entropy (KeySize). We note that, as for Bluetooth, the entropy reduction function is not reducing the length of LTK that it fixed at 16 bytes. For example, reducing the entropy of `0x123456789ABCDEF0123456789ABCDEF0` to 7 bytes generates `0x00000000000000000003456789ABCDEF0` [12, p. 2316].

BLE session keys (SK) are derived from the LTK using AES CCM [12, p. 2664]. In particular, upon secure connection establishment, the master and the slave exchange session key diversifies (SKDm and SKDs), and derive SK by encrypting the concatenation of the diversifies using AES CCM keyed with LTK. AES CCM is also used to authenticate-then-encrypt and verify-then-decrypt BLE payloads. AES CCM combines AES128 in CBC-MAC mode, and AES128 in CTR mode to provide message integrity and confidentiality [27]. To implement AES CCM, we use the python cryptographic module [7].

7.4 Evaluation of Key Negotiation Downgrade Attacks on BLE

We perform LTK negotiation downgrade attacks on 19 BLE devices from the following vendors: Google, Texas Instruments (TI), Samsung, Lenovo, Logitech, Fitbit, Xiaomi, Garmin, LG, and Motorola. For each device, we conduct our downgrade attack according to Figure 4 and by following the steps described at the end of Section 7.2. For each attack, we record the manipulated entropy negotiation over SMP in a pcapng file, and we manually verify the entropy negotiation outcome with Wireshark.

Table 4 shows our evaluation results. The first column shows the device name, the second column shows the device's OS (BLE Host), the third column shows the role of the device, and the fourth column contains the downgraded entropy value for LTK. We group the devices in two blocks. The first block includes devices supporting BLE SC (available since Bluetooth v4.2), and the second block includes devices supporting legacy BLE security (available with Bluetooth v4.0 and 4.1).

Table 4. Key Negotiation Downgrade Attack Evaluation on BLE

Device	OS (BLE Host)	Role	LTK Entropy
BLE Secure Connections (Bluetooth ≥ 4.2)			
Garmin Vivoactive 3	Proprietary	Peripheral	7 bytes
Google Pixel 2	Android	Central	7 bytes
LG K40	Android	Central	7 bytes
Samsung Gear S3	Tizen OS	Peripheral	7 bytes
Thinkpad X1 3rd	Linux	Central	7 bytes
Thinkpad X1 6rd	Linux	Central	7 bytes
TI CC1352R	TI RTOS	Central	7 bytes
BLE Legacy Security (Bluetooth 4.0 and 4.1)			
Comet Blue thermostat	Unknown	Peripheral	7 bytes
EDIFIER R1280DB speaker	Unknown	Peripheral	7 bytes
Fitbit Charge 2	Fitbit OS	Peripheral	7 bytes
ID115 HR Plus	Unknown	Peripheral	7 bytes
LG Nexus 5	Android	Central	7 bytes
Logitech MX Anywhere 2S	Nordic	Peripheral	7 bytes
Motorola G3	Android	Central	7 bytes
Samsung Galaxy J5	Android	Central	7 bytes
Samsung TV UE48J6250	Tizen OS	Peripheral	7 bytes
Xiaomi Mi band	Proprietary	Peripheral	7 bytes
Xiaomi Mi band 2 (x2)	Proprietary	Peripheral	7 bytes

The first column shows the device name, the second column shows the device OS (BLE Host), the third column shows the role of the device, and the fourth column contains the downgraded entropy value for LTK. Note that all SK derived from LTK inherit such entropy value. All 19 BLE devices that we test are vulnerable, as they accept to downgrade LTK entropy from 16 bytes to 7 bytes.

Table 4 demonstrates that all devices that we test are vulnerable to our key negotiation downgrade attack as all devices that we test accept to downgrade LTK's entropy to 7 bytes (the minimum standard-compliant value). We note that for each device, we also tested entropy values between 1 and 6 bytes and none of the devices accepted such values. The downgrade is effective regardless of the device OS, Bluetooth chip, the role, and the Bluetooth version number. Our attack affects a diverse set of devices including fitness bands, smartwatches, laptops, and IoT devices. Being an attack at the architectural level, potentially all standard-compliant BLE devices are vulnerable to our attacks.

During our evaluation, we also uncovered a problem related to BLE's strongest security mode. BLE specifies security mode 1 with level 4 as its most secure configuration (as explained in Section 2.2), and this mode mandates authenticated SC and LTK with 16 bytes of entropy [12, p. 2067]. However, we experimentally confirm that *even if a device is using security mode 1 with level 4, LTK's entropy can still be downgraded to 7 bytes*. For example, we can set LTK's entropy to 7 bytes even if the Linux kernel is compiled with BLE security mode 1, level 4 (i.e., Linux kernel BLE security level is set to `BT_SECURITY_FIPS`).

8 DISCUSSION

In this section, we discuss the root causes of the presented key negotiation downgrade attacks on Bluetooth and BLE, and we propose legacy and non-legacy compliant countermeasures.

8.1 Downgrade Attack Root Causes for Bluetooth

The root cause of the key negotiation downgrade attacks on Bluetooth lies in the Bluetooth's key negotiation specification. The Bluetooth standard defines a key negotiation protocol that allows negotiating session keys with entropy values as low as 1 byte in the clear and without message integrity. We do not see any reason to include the session key negotiation protocol in the Bluetooth standard. From our experiments (presented in Section 5.4), we observe that two devices, unless under attack, are always negotiating 16 bytes of entropy. Furthermore, the entropy reduction performed as part of the protocol does not improve runtime performances because the size of the session key is fixed to 16 bytes even when its entropy is reduced.

8.2 Bluetooth Countermeasures

In this section, we describe our proposed legacy compliant and non-legacy compliant countermeasures and we discuss the mitigation strategies that are already in place to combat our key negotiation downgrade attacks on Bluetooth. By legacy compliant, we mean countermeasures that can be implemented by individual vendors/devices without changing the standard in any way.

Legacy compliant. A legacy compliant countermeasure is to require session keys with a minimum amount of entropy that cannot be easily brute-forced. This can be accomplished by setting the minimum entropy value allowed in the Bluetooth firmware to a reasonably high value (i.e., $L_{min} = L_{max} = 16$). Another countermeasure is to automatically have the Bluetooth host (OS) check the amount of negotiated entropy after the negotiation is completed, and abort the connection if the entropy does not meet a minimum requirement. The session key entropy can be obtained by the host using the HCI Read Encryption Key Size Command. This solution requires vendors to modify the Bluetooth host, and this is the option that many vendors chose after the publication of Ref. [3]. A third countermeasure is to distrust the Bluetooth link layer and provide separate security guarantees at the application layer.

Non-legacy compliant. A non-legacy compliant countermeasure, i.e., one that violates the old standard, is to secure key negotiation using the link key (K_L). The link key is a shared and possibly authenticated secret that will be available before starting the negotiation of the session key. As the attacker must not be able to modify the victims' entropy proposals, the new key negotiation protocol must provide message integrity (and optionally confidentiality) using fresh keys derived from the link key and other parameters. An alternative countermeasure is to get rid of entropy negotiation from the specification altogether, and always use session keys with a reasonable amount of entropy, e.g., 16 bytes.

After the key negotiation attacks on Bluetooth were reported [3], several vendors, such as Apple, Google, as well as the Linux kernel, deployed dedicated mitigation strategies. Apple introduced the user notification shown in Figure 8. If the user presses Allow, then all subsequent SC with that remote device (Nexus 5 in this case) can use session keys with any entropy value (between 1 and 16 bytes). We argue that Apple mitigation is not very robust as it simply shifts the responsibility to the user. For example, a user might press the Allow button by accident, or because he/she does not understand the consequences of doing so. Google and the Linux kernel adopted the host side mitigation that we mentioned earlier in this section. Once the entropy negotiation is completed and the encryption procedure is started, the device OS (host) sends an HCI Read Encryption Key Size Command to check the entropy of the current session key. If the entropy is less than 7 bytes, the host tears down the connection. The problem with this solution is that 7 bytes of entropy is still a low value compared to 16 bytes.

Table 5 lists all the devices from our evaluation sample (presented in Section 5.4) that adopt the described mitigation strategies. To the best of our knowledge, none of the devices we have tested

Fig. 8. iPhone notification to mitigate our downgrade attacks. If the user presses Allow, then all the subsequent secure connections with the Nexus 5 can use session keys with entropy values between 1 and 16 bytes. The image is adapted from https://twitter.com/seemoolab/status/1169363042548760577/photo/1.

Table 5. Devices in Our Test Sample That Are Patched by Vendors after Our Disclosure in Ref. [3] (Before Our Disclosure, There Were No Mitigations)

Vendor	Mitigation	Min Entropy	Tested Device(s)
Apple	User notification	1 byte	iPhone X, iPad MC349LL, iPad Pro 2
Google	Host tear down	7 byte	Pixel 2, Galaxy A7, Galaxy S9
Linux	Host tear down	7 byte	Rpi 3B, 3B+, Thinkpad X1 3rd, 6th

Apple introduced a notification to the user—if the user accepts (once) the notification, then 1 byte of entropy can still be used. Google and Linux introduced code from the OS (host) side that tears down the connection if the session key entropy is lower than 7 bytes. Unfortunately, 7 bytes is still a low value.

has adopted a firmware level mitigation strategy; hence, all the chips that we have tested are still vulnerable to the downgrade attack.

8.3 Downgrade Attack Root Causes for BLE

The root cause of the key negotiation downgrade attack on BLE lies in the specification of BLE pairing. BLE pairing includes a feature exchange phase (Phase 1) where the master and slave negotiate the entropy of LTK. The standard allows negotiating entropy values as low as 7 bytes and it does not mandate to protect such negotiation. Compared to Bluetooth, BLE specifies a higher (better) minimum entropy value, i.e., 7 bytes rather than 1 byte, but such value is still too not in line with the current best practices. We do not see a strong reason to include an entropy reduction function for BLE, because, despite weakening the security of BLE, it is not improving its runtime performances as the size of LTK remains 16 bytes, even when its entropy is reduced. In our evaluation, presented in Section 7.4, there is no legitimate device taking advantage of this feature, e.g., proposing an entropy value lower than 16 bytes.

8.4 BLE Countermeasures

Legacy compliant. A legacy compliant countermeasure for BLE is to require a higher minimum entropy value for LTK. For example, the minimum entropy value can be set to 16 bytes, and BLE pairing should be aborted if the negotiated entropy (KeySize) is lower than 16 bytes. This modification requires devices to change the feature exchange phase implementation in the BLE host. For example, a Linux developer can implement this countermeasure by setting `SMP_MIN_ENC_KEY_SIZE = 16` in `net/bluetooth/smp.h`, and recompiling and reinstalling the Linux kernel. An

Table 6. Comparison with Related Work Shows the Different Security Modes that Each Paper Addresses

	Bluetooth			BLE		
Reference	Leg	SSP	SC	Leg	SC	Compromised Key
Shaked and Avishai [46]	✓	—	—	—	—	Long-term key
Haataja and Toivanen [22]	—	✓	—	—	—	Long-term key
Antonioli et al. [3]	✓	✓	✓	—	—	Session key
Ryan [43]	—	—	—	✓	—	Long-term key
This work	✓	✓	✓	✓	✓	Long-term key, session key

The security modes are Legacy (Leg), Secure Simple Pairing (SSP) and Secure Connections (SC) for Bluetooth, and Legacy (Leg) and Secure Connections (SC) for BLE. This work covers all the attack scenarios or the related papers and it is the first work exploiting BLE Secure Connections that are considered secure against eavesdropping and man in the middle attacks and directly compromising BLE session keys.

alternative countermeasure is to distrust the BLE link layer and provide the security guarantees at the application layer using protocols such as BALSA [40].

Non-legacy compliant. A non-legacy compliant countermeasure is to remove entropy negotiation from BLE pairing, and fix the entropy value to an adequate value (e.g., 16 bytes). In particular, the KeySize parameter should be removed from the feature exchange phase, and the LTK should always use 16 bytes of entropy. Alternately, the standard might include an initial key agreement phase during BLE pairing (Phase 0), where the master and the slave are establishing and authenticating a key, and then using it to protect the integrity of the feature exchange phase (including KeySize). These two countermeasures require the modification of the Bluetooth standard and the BLE host.

9 RELATED WORK

Table 6 compares the presented key negotiation downgrade attacks on Bluetooth and BLE against the relevant attacks already proposed in the literature. The first column shows the related work and the other columns contain a ✓when the attack affects Bluetooth Legacy security (Leg), Bluetooth SSP, Bluetooth SC, BLE Legacy security (Leg), and BLE SC. Shaked and Avishai [46] proposed an attack capable of breaking Bluetooth legacy security by cracking the Bluetooth PIN. Haataja and Toivanen [22] proposed two MitM attacks to break Bluetooth SSP. Antonioli et al. [3] proposed an attack capable of breaking all Bluetooth security modes by targeting session keys without having to compute the LTK and observe pairing. Ryan [43] proposed an attack capable of breaking BLE legacy security by targeting BLE legacy pairing that is insecure by design. The attacks presented in this work (last row in Table 6) not only cover all the attack scenarios of the previous work, but they are the first defeating BLE SC that are considered secure against eavesdropping and MitM attacks and directly compromising BLE session keys.

The security architecture of Bluetooth has been attacked and fixed since Bluetooth v1.0 [28, 53]. Several successful attacks on Bluetooth pairing [9, 26] have resulted in substantial revisions of the standard (e.g., the introduction of SSP and SC). Our Bluetooth attacks target the connection establishment phase and not pairing. Furthermore, the attack is effective regardless of the security guarantees provided by SSP, such as mutual user authentication, and the attacker is not required to observe SSP.

Recently, impersonation attacks on Bluetooth were proposed (BIAS) [4]. Such attacks exploit the authentication phase of Bluetooth secure connection establishment, and they can be combined with the key negotiation downgrade attacks that we propose to impersonate a device and

brute-force the Bluetooth session key without knowing the LTK. Attacks on Android, iOS, Windows, and Linux implementations of Bluetooth [5] raised concerns about the complexity of the Bluetooth standard. Our attacks are effective regardless of the implementation details.

The security of the ciphers used by Bluetooth and BLE was extensively discussed by cryptographers. For example, the SAFER+ cipher, used by Bluetooth for authentication purposes, was analyzed [32], the E_0 cipher, used by Bluetooth for encryption, was also analyzed [20], and AES CCM, used for SC, was heavily scrutinized [30]. Nevertheless, our attacks work even with perfectly secure ciphers.

Prior work about BLE legacy security (Bluetooth v4.0 and v4.1) already uncovered several flaws in the design of BLE legacy pairing. BLE's privacy mechanisms were improved [19]; BtleJack [14] was used to discover, sniff, and jam BLE connections. BtleJuice [17] was proposed as a MitM application framework for BLE. On the defensive side, BALSA [40] was proposed as an application-layer security mechanism complementary (alternative) to BLE legacy security mechanisms. The Bluetooth v4.2 standard was updated with Secure Connection to address major weaknesses of legacy BLE security. Our key negotiation downgrade attacks affect BLE devices using both legacy security and SC.

Detailed analysis of different BLE devices such as smart locks [24] and wearable devices [15] uncovered potential flaws in specific BLE's use cases. Proprietary protocols based on BLE were also analyzed for security [2] and privacy [23] issues. Specific application-layer attacks on BLE were also demonstrated [29]. Our BLE attack is agnostic to the specific use case and affects proprietary protocols building on top of BLE, and it propagates across layers (i.e., from the link layer upwards).

The most up to date survey about Bluetooth and BLE security is from NIST [42], and it recommends to use keys with 16 bytes of entropy. The survey briefly describes key negotiation and considers it as a security issue only when one of the connected devices is malicious (and not a third party). Prior surveys do not consider the problem of key negotiation at all [18] or superficially discuss it [49].

Various implementations of Bluetooth and BLE were also analyzed, and several attacks were presented on Android, iOS, Windows, and Linux implementations [5, 6]. Our downgrade attacks on Bluetooth and BLE are oblivious to the implementation details of the target platform, because if any implementation is standard-compliant, then it should be vulnerable to our attacks. For our implementation of the custom Bluetooth security procedures (presented in Section 5), we used as main references the specification [12], and third-party hardware [33] and software [36] implementations.

10 CONCLUSION

In this work, we demonstrate that Bluetooth and BLE, two different and pervasive wireless technologies, are vulnerable to standard-compliant key negotiation downgrade attacks. The attack on Bluetooth targets the secure connection establishment phase and allows downgrading the entropy of any session key to 1 byte. The attack on BLE targets BLE pairing, and it allows downgrading the entropy of any long-term and session key to 7 bytes. Keys with such lowentropy values can be brute-forced to break all the security guarantees provided by Bluetooth and BLE (e.g., the attacker can eavesdrop and decrypt all ciphertext and inject valid ciphertext in any Bluetooth and BLE network).

As our key negotiation downgrade attacks are at the architectural level, all standard-compliant Bluetooth and BLE devices are potentially vulnerable. For the same reason, our attacks are oblivious to the security modes used by the victims (e.g., SSP and SC), the Bluetooth version supported by the victims, and the manufacturers of the devices used by the victims. Our downgrade attacks can be conducted remotely, as they only require basic message manipulations, and are transparent

to the users, as the standard does not require to notify them about the negotiated entropy. Our attacks can be conducted in parallel to target more than two victims at the same time.

As the key negotiation for Bluetooth is implemented in the Bluetooth firmware of the radio chip (controller), we patch the firmware such that we can simulate a remote MitM attack who is manipulating Bluetooth's key negotiation (i.e., manipulate the LMP packets while the victims are establishing a secure connection). As the key negotiation for BLE is implemented in the main OS of the device (host), we develop a custom Linux kernel and a custom BLE user-space stack to simulate a remote MitM attacker who is manipulating BLE's key negotiation (i.e., manipulate SMP packets while the victims are pairing). We use our implementations to attack all Bluetooth and BLE devices at our disposal. All the tools that we developed for our attacks is provided as open source at https://github.com/francozappa/knob.

Our claims about the effectiveness of our attacks are confirmed by our evaluation results where we successfully attack 38 Bluetooth devices (32 Bluetooth chips) and 19 BLE devices from different manufacturers, using different versions of the protocols and different security modes. To address the serious threats related to our attacks on Bluetooth and BLE, we provide legacy and non-legacy compliant countermeasures.

REFERENCES

[1] Eman Salem Alashwali and Kasper Rasmussen. 2018. What's in a downgrade? A taxonomy of downgrade attacks in the TLS protocol and application protocols using TLS. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*. Springer, 468–487.

[2] 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)*. IEEE.

[3] 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.

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

[5] Armis Inc. 2017. The Attack Vector BlueBorne Exposes Almost Every Connected Device. Retrieved January 26, 2018 from https://armis.com/blueborne/.

[6] Armis Inc. 2018. BLEEDINGBIT Exposes Enterprise Access Points and Unmanaged Devices to Undetectable Chip Level Attack. Retrieved July 24, 2019 https://armis.com/bleedingbit/.

[7] Python Cryptographic Authority. 2019. Python cryptography. Retrieved February 4, 2019 from https://cryptography.io/en/latest/.

[8] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. 2012. Recommendation for key management part 1: General (revision 3). *NIST Special Publication* 800, 57 (2012), 1–147.

[9] Eli Biham and Lior Neumann. 2018. Breaking the Bluetooth Pairing–Fixed Coordinate Invalid Curve Attack. Retrieved October 30, 2018 from http://www.cs.technion.ac.il/ biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf.

[10] Philippe Biondi. 2018. Retrieved January 26, 2018 from Scapy: Packet crafting for Python2 and Python3. https://scapy.net/.

[11] Matt Blaze, Whitfield Diffie, Ronald L Rivest, Bruce Schneier, and Tsutomu Shimomura. 1996. *Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientists*. Technical Report. Information Assurance Technology Analysis Center, Falls Church, VA.

[12] Bluetooth SIG. 2016. Bluetooth Core Specification v5.0. Retrieved October 28, 2019 from https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043.

[13] Bluetooth SIG. 2019. Bluetooth Markets. Retrieved October 23, 2019 from https://www.bluetooth.com/markets/.

[14] Damien Cauquil. 2018. You had better secure your BLE devices. Retrieved September 27, 2019 from https://archive.org/details/youtube-VHJfd9h6G2s.

[15] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. 2018. Anatomy of a vulnerable fitness tracking system: Dissecting the fitbit cloud, app, and firmware. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 1 (2018).

[16] Arnaud Delmas. 2015. A C implementation of the Bluetooth stream cipher E0. Retrieved October 28, 2018 from https://github.com/adelmas/e0.

[17] DigitalSecurity. 2016. BtleJuice Bluetooth Smart (LE) Man-in-the-Middle framework. Retrieved July 30, 2019 from https://github.com/DigitalSecurity/btlejuice.
[18] John Dunning. 2010. Taming the blue beast: A survey of Bluetooth based threats. *IEEE Security & Privacy* 8, 2 (2010), 20–27.
[19] Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. 2016. Protecting privacy of {BLE} device users. In *Proceedings of the USENIX Security Symposium (USENIX Security)*. 1205–1221.
[20] Scott Fluhrer and Stefan Lucks. 2001. Analysis of the E0 encryption system. In *Proceedings of the International Workshop on Selected Areas in Cryptography*. Springer, 38–48.
[21] Kent Griffin, John Hastings Granbery, Hill Ferguson, David Marcus, and Michael Charles Todasco. 2015. Bluetooth low energy (ble) pre-check in. US Patent App. 14/479,200.
[22] 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.
[23] Hexway. 2019. Apple bleee. Everyone Knows What Happens on Your iPhone. Retrieved July 24, 2019 from https://hexway.io/blog/apple-bleee/.
[24] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart locks: Lessons for securing commodity Internet of Things devices. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 461–472.
[25] David Hulton. 2008. Intercepting GSM traffic. *BlackHat Briefings*.
[26] Konstantin Hypponen and Keijo M. J. 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.
[27] IETF. 2003. Counter with CBC-MAC (CCM). Retrieved October 28, 2018 from https://www.ietf.org/rfc/rfc3610.txt.
[28] Markus Jakobsson and Susanne Wetzel. 2001. Security weaknesses in Bluetooth. In *Proceedings of the Cryptographers' Track at the RSA Conference*. Springer, 176–191.
[29] Sławomir Jasek. 2016. Gattacking Bluetooth smart devices. *Black Hat USA Conference*.
[30] Jakob Jonsson. 2002. On the security of CTR+ CBC-MAC. In *Proceedings of the International Workshop on Selected Areas in Cryptography*. Springer, 76–93.
[31] Avinash Kak. 2018. BitVector.py. Retrieved October 28, 2018 from https://engineering.purdue.edu/kak/dist/BitVector-3.4.8.html.
[32] John Kelsey, Bruce Schneier, and David Wagner. 1999. Key schedule weaknesses in SAFER+. In *Proceeings of the Advanced Encryption Standard Candidate Conference*. NIST, 155–167.
[33] Paraskevas Kitsos, Nicolas Sklavos, Kyriakos Papadomanolakis, and Odysseas Koufopavlou. 2003. Hardware implementation of Bluetooth security. *IEEE Pervasive Computing* 1 (2003), 21–29.
[34] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. 2006. Breaking ciphers with COPACOBANA–A cost-optimized parallel code breaker. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 101–118.
[35] Jiun-Ren Lin, Timothy Talty, and Ozan K. Tonguz. 2015. On the potential of bluetooth low energy technology for vehicular applications. *IEEE Communications Magazine* 53, 1 (2015), 267–275.
[36] Musaria K. Mahmood, Lujain S. Abdulla, Ahmed H. Mohsin, and Hamza A. Abdullah. 2017. MATLAB implementation of 128-key length SAFER+ cipher system. *International Journal of Engineering Research and Application* 7 (2017), 49–55.
[37] 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.
[38] 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*.
[39] Yan Michalevsky, Suman Nath, and Jie Liu. 2016. MASHaBLE: Mobile applications of secret handshakes over bluetooth LE. In *Proceedings of the Annual International Conference on Mobile Computing and Networking*. ACM, 387–400.
[40] Diego A. Ortiz-Yepes. 2015. BALSA: Bluetooth low energy application layer security add-on. In *Proceedings of the International Workshop on Secure Internet of Things (SIoT)*. IEEE, 15–24.
[41] Michael Ossmann. 2019. Project Ubertooth. Retrieved October 21, 2019 from https://github.com/greatscottgadgets/ubertooth.
[42] John Padgette. 2017. Guide to bluetooth security. *NIST Special Publication* 800 (2017), 121.
[43] Mike Ryan. 2013. Bluetooth: With low energy comes low security. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, Vol. 13. USENIX, 4–4.
[44] Mike Ryan. 2015. PyBT: Hackable Bluetooth stack in Python. Retrieved June 19, 2019 from https://github.com/mikeryan/PyBT.
[45] Altaf Shaik and Ravishankar Borgaonkar. 2019. New Vulnerabilities in 5G Networks. *Black Hat USA Conference*.

[46] 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.
[47] Google Cloud Team. 2018. Google Titan Security Keys. Retrieved February 4, 2019 from https://cloud.google.com/titan-security-key/.
[48] 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. Retrieved February 4, 2019 from https://ghidra-sre.org/.
[49] Juha T. Vainio. 2000. *Bluetooth Security*. Technical Report. Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory.
[50] Mathy Vanhoef and Frank Piessens. 2017. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1313–1328.
[51] Mathy Vanhoef and Frank Piessens. 2018. Release the Kraken: New KRACKs in the 802.11 standard. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. ACM.
[52] Mathy Vanhoef and Eyal Ronen. 2020. Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd. In *Proceedings of the Symposium on Security & Privacy (SP)*. IEEE.
[53] 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.
[54] JunWeon Yoon, TaeYoung Hong, JangWon Choi, ChanYeol Park, KiBong Kim, and HeonChang Yu. 2018. Evaluation of P2P and cloud computing as platform for exhaustive key search on block ciphers. *Peer-to-Peer Network and Applications* 11 (2018), 1206–1216.
[55] Bin Yu, Lisheng Xu, and Yongxu Li. 2012. Bluetooth low energy (BLE) based mobile electrocardiogram monitoring system. In *Proceedings of the International Conference on Information and Automation*. IEEE, 763–767.

Received December 2019; revised April 2020; accepted April 2020