# BIAS: Bluetooth Impersonation AttackS

Daniele Antonioli
School of Computer and Communication Sciences
EPFL
daniele.antonioli@epfl.ch

Nils Ole Tippenhauer
CISPA Helmholtz Center
for Information Security
tippenhauer@cispa.saarland

Kasper Rasmussen
Department of Computer Science
University of Oxford
kasper.rasmussen@cs.ox.ac.uk

***Abstract*—Bluetooth (BR/EDR) is a pervasive technology for wireless communication used by billions of devices. The Bluetooth standard includes a *legacy authentication procedure* and a *secure authentication procedure*, allowing devices to authenticate to each other using a long term key. Those procedures are used during pairing and secure connection establishment to prevent impersonation attacks. In this paper, we show that the Bluetooth specification contains vulnerabilities enabling to perform impersonation attacks during secure connection establishment. Such vulnerabilities include the lack of mandatory mutual authentication, overly permissive role switching, and an authentication procedure downgrade. We describe each vulnerability in detail, and we exploit them to design, implement, and evaluate master and slave impersonation attacks on both the legacy authentication procedure and the secure authentication procedure. We refer to our attacks as Bluetooth Impersonation AttackS (BIAS).**

**Our attacks are standard compliant, and are therefore effective against any standard compliant Bluetooth device regardless the Bluetooth version, the security mode (e.g., Secure Connections), the device manufacturer, and the implementation details. Our attacks are stealthy because the Bluetooth standard does not require to notify end users about the outcome of an authentication procedure, or the lack of mutual authentication. To confirm that the BIAS attacks are practical, we successfully conduct them against 31 Bluetooth devices (28 unique Bluetooth chips) from major hardware and software vendors, implementing all the major Bluetooth versions, including Apple, Qualcomm, Intel, Cypress, Broadcom, Samsung, and CSR.**

***Index Terms*—Bluetooth, Authentication, Impersonation, Attacks, Wireless Security**

## I. Introduction

Bluetooth is a pervasive technology for low power wireless networks. Bluetooth provides Bluetooth BR/EDR and Bluetooth Low Energy (BLE) wireless stacks. In this paper we focus on *Bluetooth BR/EDR* and refer to it as Bluetooth. Bluetooth is deployed in billions of devices such as mobile phones, IoT devices, laptops, cars, medical devices and industrial devices. Bluetooth is regulated by the Bluetooth Special Interest Group (SIG), that maintains and reviews the Bluetooth standard [6]. The standard is freely available and it contains around 3000 pages. A single vulnerability in a security mechanism defined in the standard translates into billions of exploitable devices.

The Bluetooth standard specifies a security architecture that provides confidentially and integrity at the link layer [6, p. 1646]. Two Bluetooth devices that have never met before and want to establish a secure connection, first have to pair. While pairing, the devices agree upon and authenticate a long term key. This key is then used to derive session keys for subsequent secure connections. Two Bluetooth devices are expected to pair once and securely connect multiple times. During secure connection establishment the devices have to authenticate the possession of the long term key that they have established while pairing.

In a recent paper, researchers showed that Bluetooth secure connection establishment is vulnerable to man-in-the-middle attacks, even if the victims are already paired [4]. In that work however, the attack assumes that there is a legitimate secure connection to break into. The attacker cannot target isolated Bluetooth devices, because the attacker cannot prove possession of the victims' long term key during secure connection establishment.

In this paper, we demonstrate that the Bluetooth standard contains vulnerabilities enabling an attacker to impersonate a device and to establish a secure connection with a victim, without possessing the long term key shared by the impersonated device and the victim. Our attacks target the authentication phase of secure connection establishment. In particular we attack the *legacy authentication procedure* used for Legacy Secure Connections (LSC) and the *secure authentication procedure* used for Secure Connections (SC). The attacker does not have to be present when the victim and the impersonated device are paring, and does not need to observe any previous communication between them. We show how to exploit the identified vulnerabilities to mount master and slave impersonation attacks on both the legacy authentication procedure and the secure authentication procedure. Our attacks work even when the victims are using Bluetooth's strongest security modes, e.g., SSP and Secure Connections. Our attacks target the standardized Bluetooth authentication procedure, and are therefore effective against any standard compliant Bluetooth device. We refer to our attacks as **Bluetooth Impersonation Attacks (BIAS)**.

Our proof of concept implementation leverages a Bluetooth development kit (described in detail in Section VI) to send the required messages, however any device with full access to the Bluetooth firmware and a Bluetooth baseband transceiver can perform the BIAS attacks. We use our implementation to verify that the vulnerabilities in the authentication mechanisms are indeed present in real devices, and not just a quirk of the standard. We successfully attack 31 Bluetooth devices (28 unique Bluetooth chips) from major hardware and software vendors, representing all the major Bluetooth versions, including Apple, Qualcomm, Intel, Cypress, Broadcom, Samsung, and CSR.

We summarize our main contributions as follows:

- We present the BIAS attacks, the first attacks capable of bypassing Bluetooth's authentication procedure during secure connection establishment. Our attacks allow to impersonate Bluetooth master and slave devices and establish secure connections without knowing the long term key shared between the victim and the impersonated device. Our attacks exploit several flaws that we identify in the Bluetooth standard, such as lack of integrity protection, encryption, and mutual authentication. Our attacks are standard compliant, they are effective against Legacy Secure Connections and Secure Connections, and they are stealthy, i.e., no messages are show to the end user.
- We present our `BIAS` toolkit, that automates the conduction of the BIAS attacks on Bluetooth. Our toolkit configures an attack device in order to support several features, such as features impersonation, role switching, unilateral authentication, and Secure Connections downgrade.
- To demonstrate that the BIAS attacks are a serious threat, we use our implementation to successfully attack 16 Legacy Secure Connections and 15 Secure Connections devices, evaluating a total of 28 unique Bluetooth chips. Our device sample includes diverse software and hardware vendors, and all major Bluetooth versions.

We disclosed the BIAS attacks and related mitigations to the Bluetooth SIG in December 2019. The Bluetooth SIG acknowledged our findings and issued an errata to update the standard.

The rest of the paper is organized as follows: in Section II we introduce the Bluetooth stack. In Section III we present our system and adversary model and in Section IV we present our BIAS attacks on Legacy Secure Connections and Secure Connections. In Section V we discuss an alternative BIAS reflection attacks on Secure Connections. Our implementation is discussed in Section VI. We evaluate the impact and effectiveness of our attacks in Section VII and we discuss the attacks and our proposed countermeasures in Section VIII. We present the related work in Section IX. We conclude the paper in Section X.

## II. Background

### A. Bluetooth BR/EDR

Bluetooth Basic Rate Extended Data Rate (BR/EDR), referred in this section as Bluetooth, is a wireless technology for low power and short range communication, and it is the de facto standard for wireless personal area network (PAN). Bluetooth at the physical layer uses the 2.4 GHz ISM band with frequency hopping spread spectrum. Two connected Bluetooth devices hop between 79 channels at regular time interval, and each channel has a bandwidth of 1 MHz. Bluetooth allows to use adaptive frequency hopping to mitigate interference with wireless devices in range. A Bluetooth network is composed of a master device that coordinates and synchronizes up to seven slave devices. Two devices can switch master and slave roles anytime after establishing an Asynchronous Connection-Less (ACL) physical link, i.e., after baseband paging [6, p. 2100].

The specification of Bluetooth is freely available [6], and it is maintained by the Bluetooth SIG. The specification divides the Bluetooth stack into two main components the host and the controller and specifies a standard interface them, i.e., the Host Controller Interface (HCI). The standard specifies also an HCI protocol that is used by the host to send commands to the controller, and by the controller sends event to the host. The host is implemented by the device main operating system, while the controller is implemented by the firmware of the device Bluetooth chip. The standard does not provide a reference implementation for the host and the controller, and the Bluetooth vendors typically use their proprietary implementations.

The Bluetooth standard defines mechanisms to protect a Bluetooth connection at the link layer using Legacy Secure Connections procedures or Secure Connections procedures. Pairing is used by two Bluetooth devices to agree upon a long term key. The most secure and widespread pairing mechanism is Secure Simple Pairing (SSP), which uses Elliptic Curve Diffie Hellman (ECDH) for key agreement [6, p. 1691]. If the pairing devices support Secure Connections, then SSP is performed on the P-256 curve, otherwise on the P-192 curve. After pairing, and according to the security procedures in use, the devices compute a long term key from the ECDH shared secret, and they mutually authenticate such key. Pairing is performed over the air, and it uses the Link Manager Protocol (LMP).

Once two paired devices share a long term key, then they can establish multiple secure connections. Each secure connection uses a different session key, that is computed from the long term key and other public parameters. Bluetooth secure connection establishment is neither encrypted nor integrity protected, and it is used by two devices to exchange their capabilities, authenticate the long term key, compute the session key, and activate the secure connection. If the connecting devices support Secure Connections, then the secure connection establishment uses the secure authentication procedure and the connection is encrypted and integrity protected using AES CCM. Otherwise, with Legacy Secure Connections, the secure connection establishment uses the legacy authentication procedure, and the connection is encrypted using the $E_0$ stream cipher. The secure connection establishment is conducted over the air, and it uses the LMP protocol.

## III. System and Attacker Model

In this section we define our system and attacker models, as well as the notation we use in the rest of the paper.

### A. System Model

We consider two victim devices, Alice and Bob, who are using a secure Bluetooth link to communicate (see Figure 1). Note that we do not require both victims to be present at the time of the attack, we only assume that two legitimate devices exist and have communicated in the past. We assume

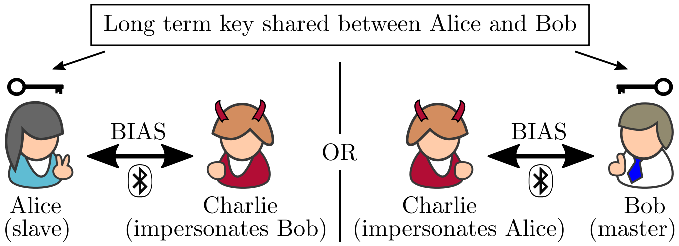


Fig. 1: Bluetooth Impersonation AttackS (BIAS) overview. Alice and Bob, the victims, share a long term key that is unknown to the attacker. The goal of the attacker (Charlie) is to establish a "secure connection" with Bob while impersonating Alice, or Alice while impersonating Bob. As Alice and Bob have different Bluetooth roles, the attacker cannot use the same impersonation attack for both.

that Alice and Bob already share a long term key, known as the link key. The key has been agreed by having completed Bluetooth's Secure Simple Pairing (either using Legacy Secure Connections or Secure Connections), and by having used a strong association model (such as passkey entry).

Without loss of generality, we assume that Bob is the Bluetooth master and Alice is the Bluetooth slave. Bob wants to establish a secure connection with Alice using the existing key described above. Conversely, Alice is willing to accept a connection from Bob using this key. Our impersonation attacks take place during secure connection establishment, that is when Alice and Bob exchange their capabilities, authenticate the long term key, compute the session key, and activate encryption. The secure connection is established either using Legacy Secure Connections or Secure Connections, according the capabilities of Alice and Bob. In both cases we assume that all the security primitives in use, such as AES and HMAC, are perfectly secure. Alice and Bob can have established previous secure connections using the long term key, although this is not a requirement.

### *B. Attacker Model*

The attacker goal is to establish a secure Bluetooth connection with Alice (or Bob), pretending to be Bob (or Alice). In Section IV we show how this can be accomplished by impersonating either Alice or Bob.

The attacker (Charlie) does not posses the long term key shared by Alice and Bob, and he does not observe them while they securely pair. Charlie is capable of eavesdropping, decoding and manipulating unencrypted packets, as well as jamming the Bluetooth spectrum. Charlie knows the public information about Alice and Bob, such as their Bluetooth names, Bluetooth addresses, protocol version numbers, and capabilities. As secure connection establishment is not encrypted, Charlie can collect Alice and Bob's characteristics by eavesdropping their communication. After the secure connection between Alice and Bob is already established, Charlie can jam the Bluetooth spectrum to force Alice and Bob to disconnect, and re-establish a secure connection.

### *C. Notation*

$K_L$ indicates the long term key resulting from pairing, know as the link key. Bluetooth authentication procedures involve challenge response protocols, and we indicate a challenge sent by a verifier with C, and a response from a prover with R. We indicate with $H_L()$ the hash function used for Legacy Secure Connections authentication, and with $H_S()$ the hash function used for Secure Connections authentication. Those functions generate R from a number of parameters, including C. We indicate the concatenation operator with $\|$ and with rand$(n)$ a function to generate $n$ random bytes. A Bluetooth address is indicated with BTADD. We use M, S, A, B and C subscripts to indicate quantities related to the master, slave, Alice, Bob and Charlie. For example, the master sends $\mathrm{C}_M$ to the slave, and the slave responds by sending $\mathrm{R}_S$ back.

## IV. Bluetooth Impersonation AttackS (BIAS)

Alice and Bob pair once to agree upon $K_L$, and then authenticate that they posses $K_L$ upon secure connection establishment using either Legacy Secure Connections or Secure Connections. The 3-tuple that uniquely identifies their secure bond is ($K_L$, $\mathrm{BTADD}_A$, $\mathrm{BTADD}_B$). When impersonating Alice or Bob, Charlie can change his Bluetooth address to $\mathrm{BTADD}_A$ or $\mathrm{BTADD}_B$, but he cannot prove the ownership of $K_L$. This is the fundamental assumption behind Bluetooth's authentication guarantees, and this assumption should protect against impersonation attacks.

In our work we present Bluetooth impersonation attacks exploiting that: i) Bluetooth secure connection establishment is neither encrypted nor integrity protected, ii) Legacy Secure Connections secure connection establishment does not require mutual authentication, iii) a Bluetooth device can perform a role switch anytime after baseband paging, iv) devices who paired using Secure Connections can use Legacy Secure Connections during secure connection establishment. As our impersonation attacks are at the architectural level of Bluetooth, they are effective against any standard compliant Bluetooth device. Our attacks are also stealthy, because the standard does not require to notify the user about (the lack of) mutual authentication and the usage of Secure Connections. We call our attacks **Bluetooth Impersonation AttackS (BIAS).**

To conduct the BIAS attacks, Charlie targets Legacy Secure Connections and Secure Connections authentication procedures during secure connection establishment. Both procedures authenticate $K_L$ using a challenge response protocol, and the procedure selection depends on Alice and Bob' supported features. The standard claims that both procedures protect secure connection establishment against impersonation attacks, as an attacker who does not know $K_L$ cannot provide a correct response to a challenge. The presented BIAS attacks on Legacy Secure Connections (see Section IV-A) and Secure Connections (see Section IV-B), demonstrate that Bluetooth secure connection establishment is vulnerable to master and slave impersonation attacks.

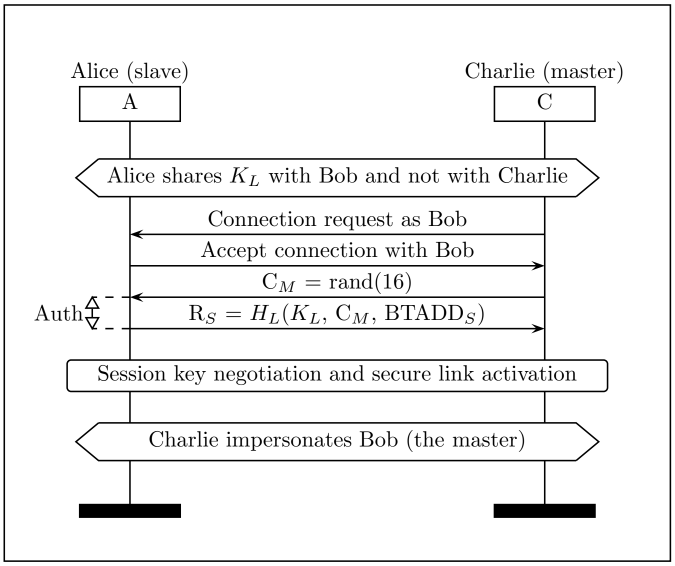

Alice (slave)
Charlie (master)
A
C
Alice shares $K_L$ with Bob and not with Charlie
Connection request as Bob
Accept connection with Bob
$\mathrm{C}_M$ = rand(16)
Auth
$\mathrm{R}_S = H_L(K_L, \mathrm{C}_M, \mathrm{BTADD}_S)$
Session key negotiation and secure link activation
Charlie impersonates Bob (the master)

Fig. 2: BIAS master impersonation attack on Bluetooth legacy authentication procedure. Charlie establishes a connection with Alice pretending to be Bob. Charlie sends $\mathrm{C}_M$ to Alice, gets $\mathrm{R}_S$ from Alice. As the Bluetooth standard does not mandate to use the legacy authentication procedure mutually while establishing a secure connection, Alice does not have to authenticate that Charlie knows $K_L$.

### A. BIAS Attacks on Legacy Secure Connections

Anytime Alice (slave) and Bob (master) want to establish a secure connection they use a procedure to authenticate $K_L$, and the standard defines such procedure as *legacy authentication procedure [6, p. 558]*. The procedure is described in Figure 10 and works as follows. The master computes and sends $\mathrm{C}_M$ to the slave. The slave computes the response $R_S = H_L(K_L, C_M, \mathrm{BTADD}_S)$, and sends it to the master. The master then computes a response using the same function with the same inputs, and compares it against $\mathrm{R}_S$. If the values are equal, then the master concludes that he is sharing the same $K_L$ with the slave.

The legacy authentication procedure provides unilateral authentication. When Alice and Bob are pairing such procedure is used two times to achieve mutual authentication, i.e., Alice authenticates Bob and then Bob authenticates Alice. A central issue is that *the Bluetooth standard does not require to use the legacy authentication procedure mutually during secure connection establishment*, see [6, p. 559] and [6, p. 1671]. From our experiments, presented in Section VII, we confirm that all Legacy Secure Connections devices that we tested are using the legacy authentication procedure unilaterally during secure connection establishment as only the master authenticates the slave. Thus, if Charlie can impersonate the master, then he can complete secure connection establishment without having to authenticate to the slave.

Charlie impersonates Bob (master), and completes the secure connection establishment with Alice as described in Figure 2. Charlie requests a connection to Alice pretending to be Bob, and Alice accepts the connection. Charlie connects as Bob by forging Bob's addresses and capabilities that are public. Charlie sends $\mathrm{C}_M$ to Alice, and Alice computes $\mathrm{R}_S$ based on $K_L$, $\mathrm{C}_M$, and $\mathrm{BTADD}_S$, and sends $\mathrm{R}_S$ to Charlie. Then, Charlie completes the session key negotiation and secure link activation as Bob, without having to prove he owns $K_L$ to Alice.

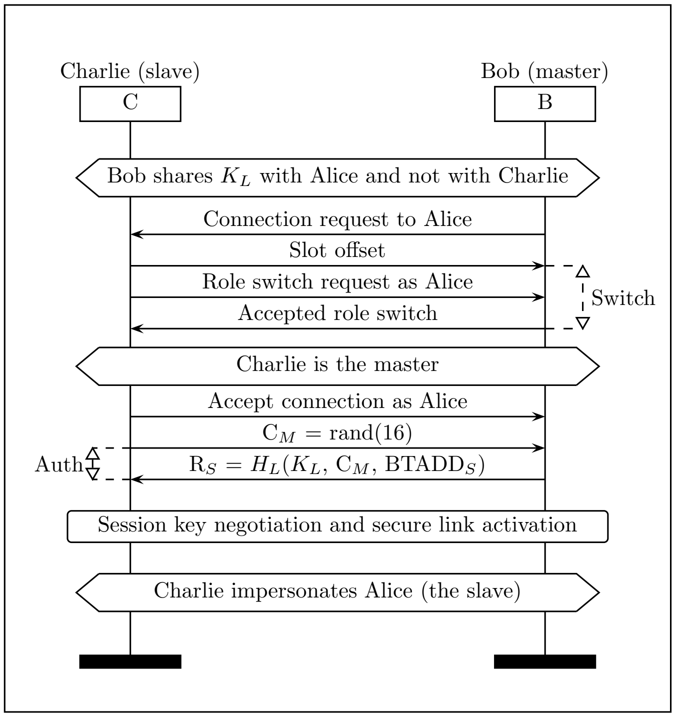

Charlie (slave)
Bob (master)
C
B
Bob shares $K_L$ with Alice and not with Charlie
Connection request to Alice
Slot offset
Role switch request as Alice
Accepted role switch
Switch
Charlie is the master
Accept connection as Alice
$\mathrm{C}_M$ = rand(16)
Auth
$\mathrm{R}_S = H_L(K_L, \mathrm{C}_M, \mathrm{BTADD}_S)$
Session key negotiation and secure link activation
Charlie impersonates Alice (the slave)

Fig. 3: BIAS slave impersonation attack on Bluetooth legacy authentication procedure. Bob requests a connection to Charlie who is impersonating Alice, and Charlie requests a role switch before accepting the connection request. Bob accepts the role switch and becomes the network slave (prover). Charlie, the network master (verifier), sends $\mathrm{C}_M$ to Bob, and Bob authenticates to Charlie by sending him $\mathrm{R}_S$. Bob does not have to authenticate that Charlie knows $K_L$.

Charlie can also impersonate the slave by maliciously taking advantage of Bluetooth's role switch procedure. Bluetooth uses a master slave medium access protocol, to keep the master and the slave synchronized. The standard specifies that the master and slave roles can be switched anytime after baseband paging is completed [6, p. 595]. This is problematic because Charlie can use this to impersonate the slave device by initiating a role switch and become the master (verifier) before the unilateral authentication procedure is started, and then complete the secure connection establishment without having to authenticate. In our experiments we are able to reliably perform this adversarial role switch for all except one of our tested devices (more details in Section VII). This feature of Bluetooth was never investigated in a security context, and is thus an entirely novel attack technique.

Charlie impersonates Alice (slave), and completes the secure connection establishment with Bob as in Figure 3. Bob requests a connection to Charlie (posing as Alice). Charlie sends a slot offset synchronization packet and then a role switch request to Bob. The role switch procedure is not authenticated, but Bob has to accept the request to be standard compliant. Charlie (the new master), accepts the connection and immediately starts the unilateral legacy authentication procedure by sending $\mathrm{C}_M$ to Bob. Bob authenticates to Charlie by sending him $\mathrm{R}_S$. Then, Charlie completes the session key negotiation and secure link activation as Alice, without having to authenticate $K_L$ to Bob.

In summary, Charlie is capable of impersonating both the master and slave for every possible usage of unilateral legacy authentication. The root problems are that the specification of Bluetooth Legacy Secure Connections does not mandate mutual authentication for secure connection establishment and that role switch is allowed anytime after baseband paging. From our experiments we see that the legacy authentication procedure is used mutually while pairing, e.g., Alice authenticates to Bob first and then Bob authenticates to Alice. This does not protect against our impersonation attacks, as they are conducted during secure connection establishment and not during pairing.

### *B. BIAS Downgrade Attacks on Secure Connections*

In this section we describe how Charlie can impersonate Secure Connections devices using standard compliant downgrade attacks. Secure Connections uses stronger cryptographic primitives than Legacy Security Connections, and is considered the most secure way to pair and establish secure connections. All Secure Connections devices that we test are vulnerable to our downgrade attacks (see Section VII).

Secure Connections provides a mutual authentication procedure, known in the standard as the *secure authentication procedure* [6, p. 559]. The procedure is described in Figure 11 and works as follows. Alice (slave) and Bob (master) exchange $\mathrm{C}_S$ and $\mathrm{C}_M$ in no particular order. Both then compute

$$R_M \| R_S = H_S(K_L, \mathrm{BTADD}_M, \mathrm{BTADD}_S, C_M, C_S),$$

using the $H_S()$ hash function. The Bluetooth standard is not quite clear regarding the order of the responses. In [6, p. 559] the slave should send $\mathrm{R}_S$ first, but in [6, p. 1673] the master sends $\mathrm{R}_M$ first. In our experiments, the slave always sends $\mathrm{R}_S$ first and we adopt this convention. After the mutual computation of the responses, Alice sends $\mathrm{R}_S$ to Bob and Bob sends $\mathrm{R}_M$ to Alice. Alice and Bob verify that the responses that they get match the ones that they compute. If the both verifications are successful then $K_L$ is mutually authenticated.

Our BIAS attack on Secure Connections is enabled by a downgrade vulnerability in the specification of Secure Connections. In particular, the Bluetooth standard *does not require two devices that used Secure Connections for pairing to always use Secure Connections for secure connection establishment, and does not protect the negotiation of Secure Connections*. In other words, Alice and Bob, even if they support and they already paired using Secure Connections, can establish secure connections using Legacy Secure Connections.

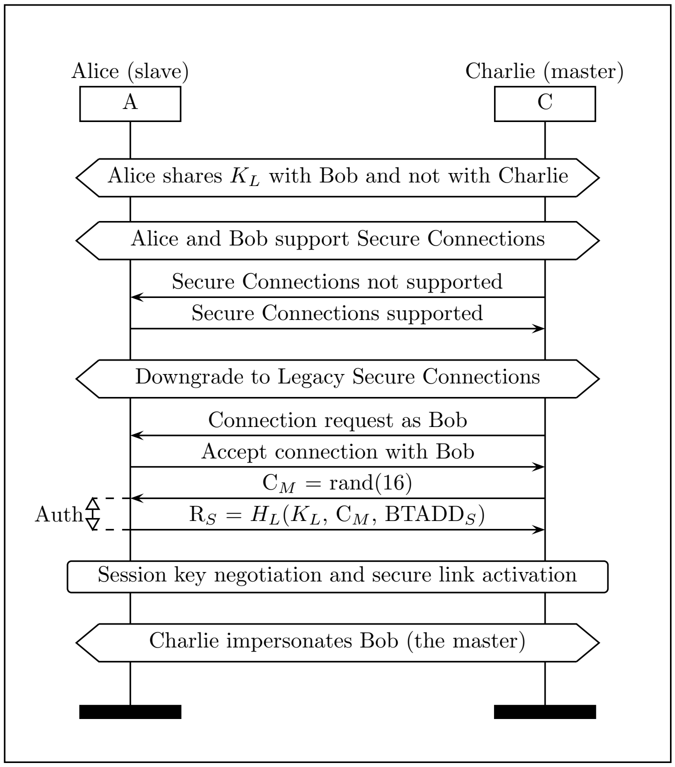

Alice (slave)
A
Charlie (master)
C
Alice shares $K_L$ with Bob and not with Charlie
Alice and Bob support Secure Connections
Secure Connections not supported
Secure Connections supported
Downgrade to Legacy Secure Connections
Connection request as Bob
Accept connection with Bob
$\mathrm{C}_M$ = rand(16)
Auth
$\mathrm{R}_S = H_L(K_L, \mathrm{C}_M, \mathrm{BTADD}_S)$
Session key negotiation and secure link activation
Charlie impersonates Bob (the master)

Fig. 4: BIAS master impersonation downgrade attack on Bluetooth secure authentication procedure. Secure connection establishment, including feature exchange, is neither encrypted nor integrity protected. During feature exchange Charlie, as Bob, declares that Secure Connections is not supported and Alice declares that Secure Connections is supported. The secure connection establishment is downgraded to Legacy Secure Connections. Charlie establishes a connection with Alice as Bob. Charlie, sends $\mathrm{C}_M$ to Alice, and gets $\mathrm{R}_S$ from Alice. Charlie starts the session key negotiation without having to authenticate to Alice.

Charlie takes advantage of these vulnerabilities to pretend that the impersonated device (either Alice or Bob) does not support Secure Connections to downgrade secure connection establishment with the victim to Legacy Secure Connections. As a result of the downgrade, Charlie and the victim use the legacy authentication procedure rather than the secure authentication procedure, and Charlie can bypass secure connection establishment authentication as in IV-A. In the following paragraphs we describe the master and slave downgrade attacks on Secure Connections in detail.

Assuming that Alice and Bob have already paired and they support Secure Connections, then Charlie impersonates Bob (master) as described in Figure 4. During the feature exchange phase Charlie, pretending to be Bob, tells Alice that Secure Connections is not supported. Even if Alice tells Charlie that she does support Secure Connections, secure connection establishment is downgraded to Legacy Secure Connections. Then, Charlie establishes a connection with Alice as Bob.

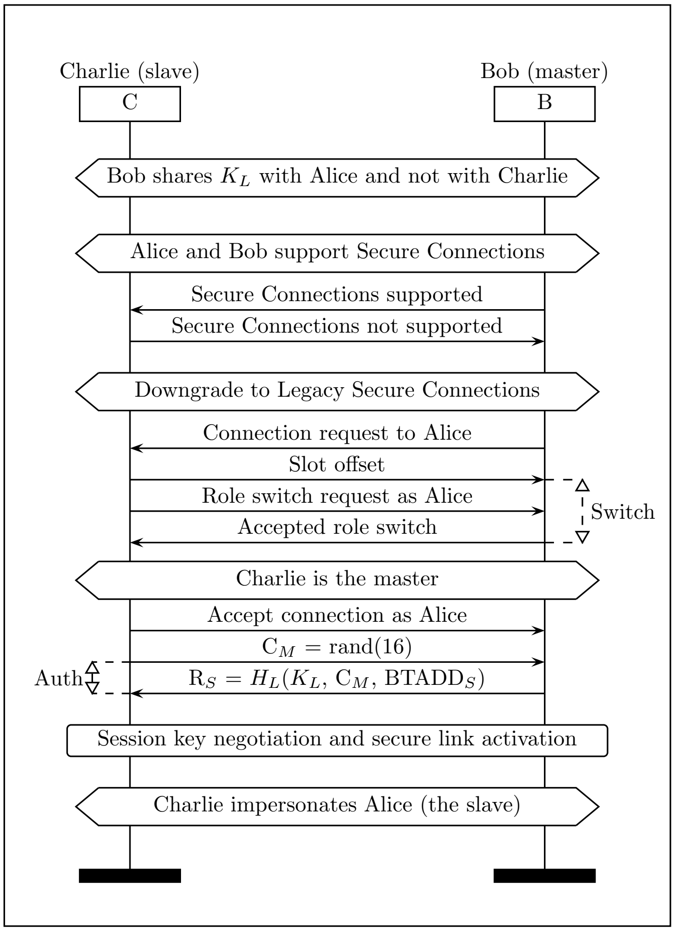

Charlie (slave)
Bob (master)
C
B
Bob shares $K_L$ with Alice and not with Charlie
Alice and Bob support Secure Connections
Secure Connections supported
Secure Connections not supported
Downgrade to Legacy Secure Connections
Connection request to Alice
Slot offset
Role switch request as Alice
Accepted role switch
Switch
Charlie is the master
Accept connection as Alice
$C_M$ = rand(16)
Auth
$R_S = H_L(K_L, C_M, BTADD_S)$
Session key negotiation and secure link activation
Charlie impersonates Alice (the slave)

Fig. 5: BIAS slave impersonation downgrade attack on Bluetooth secure authentication procedure. Secure connection establishment, including feature exchange, is neither encrypted nor integrity protected. During feature exchange Charlie, as Alice, declares that Secure Connections is not supported and Bob declares that Secure Connections is supported. The secure connection establishment is downgraded to Legacy Secure Connections. Bob establishes a connection with Charlie. Charlie, sends $C_M$ to Bob, and gets $R_S$ from Bob. Charlie starts the session key negotiation without having to authenticate to Bob.

Charlie, being the only verifier, performs unilateral legacy authentication, and he establishes a secure connection without having to authenticate to Alice.

Charlie impersonates Alice (slave) as described in Figure 5. During the feature exchange phase, Bob tells Alice that he supports Secure Connections. Charlie as Alice, tells Bob that he does not support Secure Connections. The secure connection establishment is downgraded to Legacy Secure Connections. Bob sends a connection request to Alice and Charlie performs a role switch and becomes the master, before accepting the connection request. Charlie, being the only verifier, performs unilateral legacy authentication, and he establishes a secure connection without having to authenticate to Bob.

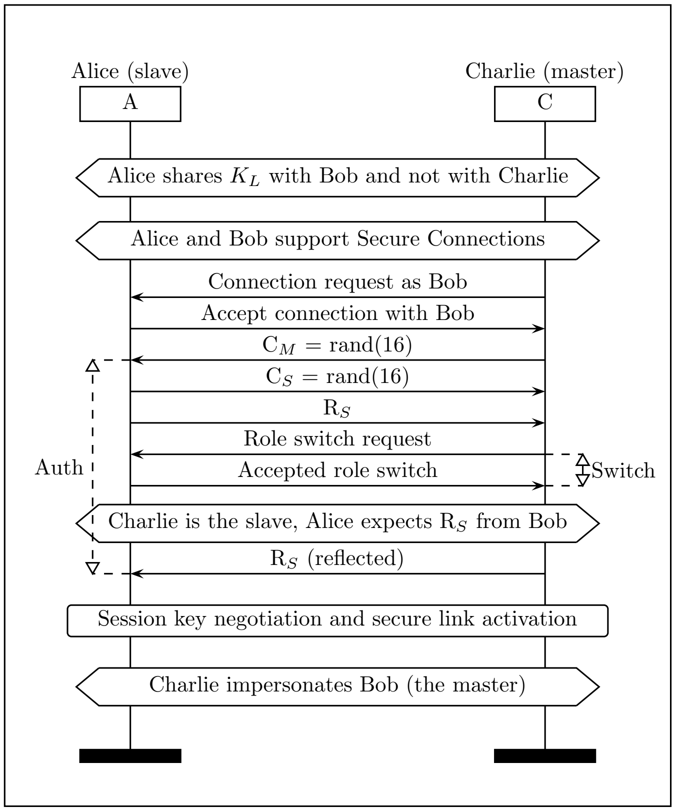

Alice (slave)
Charlie (master)
A
C
Alice shares $K_L$ with Bob and not with Charlie
Alice and Bob support Secure Connections
Connection request as Bob
Accept connection with Bob
$C_M$ = rand(16)
$C_S$ = rand(16)
$R_S$
Role switch request
Accepted role switch
Switch
Auth
Charlie is the slave, Alice expects $R_S$ from Bob
$R_S$ (reflected)
Session key negotiation and secure link activation
Charlie impersonates Bob (the master)

Fig. 6: BIAS master impersonation reflection attack on Bluetooth secure authentication procedure. Charlie establishes a connection with Alice, pretending to be Bob. Charlie sends $C_M$ to Alice, and Alice sends $C_S$ to Charlie. Alice computes $R_M$ and $R_S$ and sends $R_S$ to Charlie. Charlie performs a role switch, Alice becomes the master and expects $R_S$ from Charlie. Charlie reflects $R_S$ to Alice and completes the secure authentication procedure without possessing $K_L$.

## V. BIAS Reflection Attacks on Secure Connections

We now present another (alternative) way to attack Secure Connections authentication using *reflection attacks*. In a reflection attack, the attacker tricks the victim into answering his own challenge and giving the response to the attacker. The attacker then authenticates to the victim by reflecting (sending back) the response. We note that while we interpret the standard as not protecting against reflection attacks, we do not present an implementation as part of this work.

Our reflection attacks assume that Charlie is able to switch role during the secure authentication procedure after receiving a response from the remote victim. The Bluetooth standard allows role switching anytime after baseband paging [6, p. 595], but it is not clear about the possibility to role switch in the middle of an authentication procedure. In the rest of this section we describe what would happen if this is the case. In the following, we assume that the slave always sends R first as in [6, p. 559].

Figure 6 describes how Charlie reflects $R_S$ back to Alice (slave), while impersonating Bob (master). Charlie sends a connection request to Alice pretending to be Bob, and

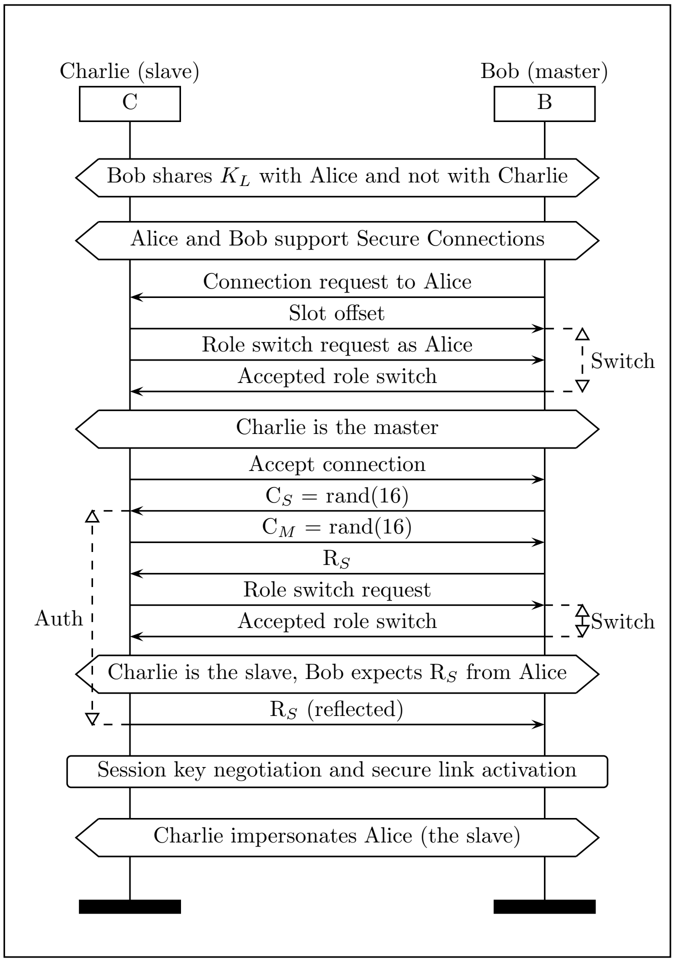

Charlie (slave)
C
Bob (master)
B
Bob shares $K_L$ with Alice and not with Charlie
Alice and Bob support Secure Connections
Connection request to Alice
Slot offset
Role switch request as Alice
Accepted role switch
Switch
Charlie is the master
Accept connection
$C_S$ = rand(16)
$C_M$ = rand(16)
$R_S$
Role switch request
Accepted role switch
Switch
Auth
Charlie is the slave, Bob expects $R_S$ from Alice
$R_S$ (reflected)
Session key negotiation and secure link activation
Charlie impersonates Alice (the slave)

Fig. 7: BIAS slave impersonation reflection attack on Bluetooth secure authentication procedure. Bob sends a connection request to Alice. Charlie, who impersonates Alice, requests a role switch before accepting the connection request. Bob accepts the role switch and Charlie becomes the master and connects with Bob. Charlie sends $C_M$ to Bob, and Bob sends $C_S$ to Charlie. Bob computes $R_M$ and $R_S$ and sends $R_S$ to Charlie. Charlie becomes the slave by performing another role switch, reflects $R_S$ to Bob and completes the secure authentication procedure without possessing $K_L$.

Alice accepts the connection. Charlie sends $C_M$ to Alice, and Alice sends $C_S$ to Charlie. The values and the ordering of the challenges do not influence the effectiveness of our attacks. Alice computes $R_M$ and $R_S$ using $H_S$ as described in Section IV-B, but Charlie cannot compute such responses because he does not know $K_L$. Right after Alice sends $R_S$ to Charlie, Charlie sends a role switch request to Alice, Alice accepts the role switch and Charlie becomes the new slave. Now Alice, the new master, expects $R_S$ from Charlie, thus Charlie reflects $R_S$ to Alice, and completes the secure (mutual) authentication procedure without knowing $K_L$.

Charlie can also reflect $R_S$ back to Bob (master) while impersonating Alice (slave) as described in Figure 7. This reflection attack uses the same logic of the master impersonation reflection attack that we just described. What changes is the first step of the attack, where Charlie has to pretend to be Alice (the slave) and he has to perform an extra role switch to become the master before accepting the connection with Bob.

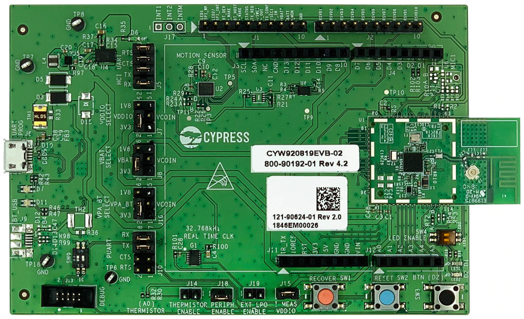


Fig. 8: The CYW920819EVB-02 development board from Cypress. The board includes a CYW20819 SoC (inside the dashed white square region). The SoC implements Bluetooth 5.0, supports Secure Connections, and is managed over USB.

The Bluetooth standard mentions that "the usage of $\mathrm{BTADD}_M$ and $\mathrm{BTADD}_S$ in the secure authentication procedure ($H_S$) prevents a simple reflection attack" [6, p. 1672]. Using unique identifiers as part of a challenge response protocol is a common reflection attack countermeasure. However, it is not enough in this case because roles can be switched before and after the responses are computed and sent.

The standard has a footnote in the specification of the legacy authentication procedure saying "The reflection attack actually forms no threat because all service requests are dealt with on a FIFO basis. When preemption is introduced, this attack is potentially dangerous." [6, p. 1671]. This is a reasonable warning, but it should be extended to the secure authentication procedure, and it does not prevent to use a priority queue instead of a FIFO to manage services. Overall, we argue that the standard should include our reflection attacks in the threat model and explicitly disallow role switching during the secure authentication procedure.

## VI. Implementation

We implement the BIAS attacks presented in Sections IV-A and Section IV-B using a CYW920819EVB-02 evaluation board and a Linux laptop. In this section we describe the board, and the relevant information about its Bluetooth firmware that we reverse engineered. Then, we present our `BIAS` toolkit, that is the first toolkit that automates the impersonation of arbitrary Bluetooth devices and the setup needed to conduct the BIAS attacks. Finally, we explain how we use the `BIAS` toolkit to implement master and slave impersonation attacks on Legacy Secure Connections and Secure Connections.

### A. CYW920819EVB-02 Bluetooth Development Board

To implement the BIAS attacks we use the CYW920819EVB-02 development board [9] (in Figure 8) and a Linux laptop. The board includes a CYW20819 SoC, that implements Bluetooth

5.0 and supports Secure Connections [8]. The CYW20819 main core is an ARM Cortex M4 clocked at 96 MHz and using the ARMv7E-M architecture. The board provides access via USB to the HCI UART bus, used to interface a Bluetooth host (e.g., our laptop) with the board Bluetooth controller, and the HCI peripheral bus used for logging and debugging. The board has a JTAG interface that we use for hardware level debugging with a J-Link EDU debug probe [28]. We program the board cross-compiling the code from our laptop using the libraries, drivers and tools provided by the ModusToolbox SKD [10].

The board stores the Bluetooth firmware in 1 MB of read-only ROM, and the Bluetooth application in a 256 KB on-chip flash that is readable, writable and executable. The board has a 176 KB on-chip RAM, and executing the application from flash allows to save RAM space with minimal memory latency overhead. We write application code on our laptop, cross compile it for the board, and load it via USB on the board's flash memory. Loading the application code is referred in the board's documentation as "re-flashing the firmware", but what is re-flashed is the application code, as the Bluetooth firmware cannot be re-flashed.

### B. Reverse Engineering the Board Bluetooth Firmware

The implementation of our BIAS attacks requires to modify the board's Bluetooth firmware as the firmware implements authentication and secure session establishment procedures. Unfortunately, the board SDK does not include the firmware source code, and the capability to re-flash a modified firmware. However, we find that the SDK contains the firmware debugging symbols, and supports proprietary HCI commands to read and write the board's RAM [7]. We use the proprietary read RAM command to dump the RAM content into a binary blob at runtime. Then we find a Makefile containing the memory layout information, and using such information we extract several regions from the binary blob including ROM, RAM, and patch RAM.

The ROM region contains the firmware code, the RAM region contains the runtime memory, including the stack and the heap, and the patch RAM contains a table of patches that are applied after boot using a proprietary patching mechanism from Cypress, known as "PatchRom". PatchRom allows patch the firmware without having to change the ROM by redirecting code from ROM at runtime to patches in RAM. Patching slots have to be used wisely as the board only has 16 slots.

To reverse engineer the board's Bluetooth firmware, we load the dumped ROM, the symbols, the RAM, and the patch RAM regions into a Ghidra project [32]. Ghidra is an open source disassembler and decompiler developed by the US National Security Agency (NSA), compatible with ARM binaries. We configure the Ghidra project to use the ARM Cortex M4 architecture in thumb mode and we perform a first pass of Ghidra automatic analysis. We spent a considerable amount of time reverse engineering the firmware to uncover its main components, data structures, and control flow blocks. Those information are essential to develop correct firmware patches to implement the BIAS attacks. In the remaining we describe the most relevant findings about the board Bluetooth firmware.

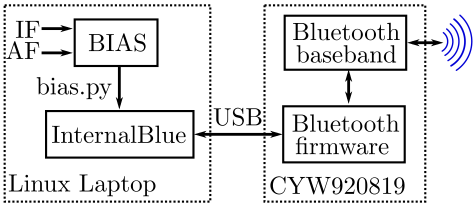


Fig. 9: Our BIAS Toolkit. The toolkit takes as inputs an Impersonation File (IF) containing information about the device to be impersonated and an Attack File (AF) that contains information about the laptop and the board. The toolkit, among others, produces bias.py that can be used with InternalBlue [23] to setup our CYW920819EVB-02 to conduct the BIAS attacks.

The firmware has a standard ARM Cortex M4 memory layout and interrupt vector table, and the entry point is the reset interrupt handler. The firmware boots using slimboot, initialize the RAM and the peripherals, and then loads the OS kernel. The OS is ThreadX, a proprietary real time operating systems (RTOS) [21]. The firmware execution environment is multi-threaded, and it is managed by a real-time and priority-based scheduler.

The firmware implements the LMP protocol using an API based on tasks. Whenever an LMP packet is received or has to be sent, a specific task is created according to the type of LMP packet. Each type of packet has a callback, and the callbacks are stored in a table in RAM. A LMP dispatcher calls the appropriate callback according to the type of LMP packet. HCI packets are handled using the same logic.

Interrupts are used to communicate between the SoC main ARM core and the peripheral devices. The firmware interfaces with a security module peripheral to accelerate in hardware the computations of standard security primitives, such as AES and SHA-1. The firmware does not use address space layout randomization, data execution prevention and stack canaries, indeed we can perform arbitrary control flow manipulations on the firmware.

### C. Our BIAS Toolkit

After collecting enough information by reverse engineering the board firmware, we developed the BIAS toolkit to automate our BIAS attacks. To the best of our knowledge, the BIAS toolkit is the first implementing Bluetooth impersonation attacks. We plan to release the toolkit as open source at https://github.com/francozappa/bias. In our experiments, presented in Section VII, we use our toolkit to successfully attack 31 Bluetooth device (28 unique Bluetooth chips).

Figure 9 presents an high level description of the BIAS toolkit. The toolkit takes as inputs the Impersonation File (IF) and the Attack File (AF). The Impersonation File contains information about the device that we impersonate, such as Bluetooth address, Bluetooth name, and Secure Connections

| Feature | Description |
|---|---|
| Impersonation | Impersonate Bluetooth address, name, version, chipset, device class, and features. |
| Role Switch 1 | Role switch before accepting a connection request from a master. |
| Role Switch 2 | Role switch at any point in time that is legal in the specification. |
| Secure Connections | Enable either Legacy Secure Connections or Secure Connections. |
| No Authentication | Ignore authentication requests and missing link keys. |
| Link Key Mgmt | Read, write and delete link keys from the firmware. |
| Logging | Enable Link Manager (LM) logging on the board, including LMP packets. |
| KNOB Attack | Downgrade the entropy of the Bluetooth session keys as in [4]. |

TABLE I: `BIAS` toolkit main features. The `BIAS` toolkit allows the attack device to impersonate a victim, perform arbitrary role switches, disable Secure Connections, ignore authentication requests, manage link keys, log LMP packets, and chain BIAS and KNOB attacks.

| | Nexus 5 | Pixel 2 |
|---|---|---|
| Bluetooth Address | CC:FA:00:70:DC:B6 | 40:4E:36:A8:BF:5F |
| Bluetooth Name | nex | pixel2 |
| Core specification | 4.1 | 5.0 |
| Chip vendor | Broadcom | Qualcomm |
| Chip subversion | 24841 | 702 |
| Device class | `0c025a` | `0c025a` |
| Feature page 0 | `bffecffedbff7b8f` | `fffe8ffed83f5b87` |
| Feature page 1 | `0700000000000000` | `0f00000000000000` |
| Feature page 2 | `3008000000000000` | `4503000000000000` |
| IO mask | `01` | `01` |
| OOB mask | `00` | `00` |
| AuthReq mask | `03` | `03` |
| Secure Connections | False | True |

TABLE II: We use our `BIAS` toolkit to impersonate, among others, a Nexus 5 and a Pixel 2. The Nexus 5 supports Legacy Secure Connections, and the Pixel 2 supports Secure Connections.

support. The Attack File contains information about the attack device such as the name of the HCI interface used by our laptop, and the addresses of the functions that we want to patch in the board Bluetooth firmware.

The toolkit, given `IF` and `AF`, produces a `bias.py` Python script that can be used with InternalBlue [23]. InternalBlue is an open source toolkit that provides a Python API to interact with Cypress Bluetooth chips, including the CYW20819 used by our board. Using a shell script and `bias.py` we transform our attack device into the impersonated device, and we configure the attack device to perform the BIAS attacks. The implementation details of our attacks are presented in Section VI-D (Legacy Secure Connections) and Section VI-E (Secure Connections).

The main features of the `BIAS` toolkit are summarized in Table I. The Impersonation feature allows to modify the attack device such that it impersonates the victim specified in `IF`. The Role Switch features allow the attack device to perform master and slave role switches in different scenarios, e.g., when starting secure connection establishment. The Secure Connections feature enables or disables Secure Connections for the attack device and is used in the Secure Connections downgrade attack. The No Authentication feature allows the attack device to ignore remote authentication requests and missing link keys, and is used to exploit unilateral Legacy Secure Connections authentication. The Link Key Mgmt feature enables to read, write, and delete link keys from the attack device. The Logging feature enables Link Manager logging by sending a vendor specific HCI command to the board. The KNOB attack feature allows to downgrade the entropy of the session key as in [4], after completing a BIAS attack.

Our `BIAS` toolkit takes advantage of InternalBlue Python API. We use `sendHciCommand(opcode, args)` to send HCI commands from the laptop to the board, including the Cypress proprietary ones. With this capability we are able, among others, to change the Bluetooth address of the board. We write and read the firmware RAM using `writeMem(address, value)` and `readMem(address, bytes)`. Those capabilities allow to change, among others, the board Bluetooth name and Secure Connections support and to write our patches in RAM. InternalBlue internally uses pwntools, and we use pwntools's `asm(code, vma)` to create our patches before writing them to the firmware RAM. We use `patchRom(address, asmbranch)` to patch the board firmware such that once the firmware execution reaches `address` it executes `asmbranch` which in turn jumps to the address of one of our patches in RAM. The firmware patching capability allows, among others, to perform adversarial role switches and unilateral legacy authentication.

### *D. BIAS Implementation for Legacy Secure Connections*

The master and slave BIAS attacks on Legacy Secure Connections take advantage of unilateral legacy authentication and the adversarial role switch as described in Section IV-A. To implement such attacks, our attack device needs the following capabilities. The attack device has to impersonate a Bluetooth device that supports Legacy Secure Connections, has to switch role before accepting a connection from a master, and has to ignore authentication requests from the remote victim, if any, and perform the standard compliant unilateral authentication procedure. We now describe how we implement such capabilities on our attack device consisting of a CYW920819EVB-02 board connected to a Linux laptop (as in Figure 9).

In this section we use as a reference example the impersonation of a Nexus 5 smartphone. The Nexus 5 runs Android 6.0.1, and includes a CYW4339 Bluetooth 4.1 SoC. Using our `BIAS` toolkit, we select the Nexus 5 Impersonation File (`IF`) from our database and we configure our attack device to impersonate all the capabilities listed in the left column of of Table II. As a result, a user discovering Bluetooth devices cannot tell our attack device apart from our Nexus 5 as they advertise the same capabilities with the same identifiers.

Then we use our toolkit to configure the adversarial role switch and the unilateral authentication for the board, taking advantage of the Attack File (`AF`) that we provide. The role switch is configured by patching the board firmware. We hook the function that handles remote connection requests and we patch it such that our board always role switch from slave to master before accepting a connection request. Unilateral legacy authentication is enforced by two more patches to the board firmware. The first patch immediately starts the legacy authentication procedure after a connection is established. The second patch immediately starts the session key negotiation procedure after the board authenticates the remote victim. As a result, our attack device, while impersonating the Nexus 5, does not have to authenticate during secure session establishment regardless of its Bluetooth role.

$$R_P = E_1(K_L, \mathrm{BTADD}_P, C_V) \tag{1}$$

To validate the responses produced by the legacy authentication procedure we implement the $H_L$ hash function introduced in Section IV-A. Such hash function uses $E_1$ to compute the response ($R_P$) from the link key ($K_L$), the Bluetooth address of the prover (BTADD$_P$), and the challenge sent by the verifier (C$_V$) as in Equation 1. $E_1$ internally uses the SAFER+ block cipher and we implement it as well. To check the correctness of our implementation we successfully test it against the test vectors in the standard [6, p. 1620].

### E. BIAS Implementation for Secure Connections

In this section we present the implementation of the Secure Connections downgrade attack that we describe in Section IV-B. Such attack requires the following capabilities. The attack device has to impersonate a device that supports Secure Connections but downgrade to Legacy Secure Connections during secure connection establishment. The attack device has to switch role before accepting a connection from a master device, and perform the standard compliant unilateral authentication. We now describe how we implement those capabilities on our attack device consisting of the CYW920819EVB-02 board connected to a Linux laptop (as in Figure 9).

In this section we use as a reference example the impersonation of a Pixel 2 smartphone. The Pixel 2 runs Android 10, and includes the Snapdragon 835 Bluetooth 5.0 SoC. Using our `BIAS` toolkit, we select the Pixel 2 Impersonation File (`IF`) from our database and we configure our attack device to impersonate all the capabilities listed in the right column of of Table II. As a result, a user discovering Bluetooth devices cannot tell our attack device apart from our Pixel 2 as they advertise the same capabilities with the same identifiers.

Then we use our toolkit to configure the Secure Connections downgrade, adversarial role switch and unilateral authentication for the board via the related `AF`. The Secure Connections downgrade is implemented using a patch that modifies the Secure Connections support flags in the board Bluetooth firmware. Adversarial role switch and unilateral authentication are implemented using the same patches that we describe in Section VI-D. As a result, our attack board, while impersonating the Pixel 2, downgrades the authentication procedure used for secure connection establishment and bypasses authentication.

$$K_A = h_4(K_L, \text{"btdk"}, \mathrm{BTADD}_M, \mathrm{BTADD}_S) \tag{2}$$

$$R_M \| R_S = h_5(K_A, C_M, C_S) \tag{3}$$

To validate the responses produced by the secure authentication procedure while the victims are pairing we implement the $H_S$ hash function presented in Section IV-B. The hash function internally uses $h_4$ as in Equation 2 to compute a device authentication key ($K_A$) from $K_L$, the "btdk" string, BTADD$_M$ and BTADD$_S$. Then, $K_A$, $C_M$ and $C_S$ are used by $h_5$ to compute the concatenation of $R_M$ and $R_S$ as in Equation 3. We implement $h_4$ and $h_5$ following their specification [6, p. 1699], and we test our implementation using the test vectors provided in the standard [6, p. 1615].

## VII. Evaluation

In this section we describe our BIAS attacks evaluation setup and results. We successfully conducted master and slave impersonation attacks on 16 Legacy Secure Connections devices and on 15 Secure Connections devices, using a total of 28 unique Bluetooth chips.

### A. Evaluation Setup

We consider an attack scenario with victim A, victim B, and the attacker. Victim A and the attack device are two CYW920819EVB-02 development boards connected to two laptops running Linux, supporting Secure Connections. Victim B is any other Bluetooth device at our disposal, and it might support Secure Connections. Victim A is paired with victim B, and the attacker does not know their long term key ($K_L$). The attacker impersonates victim A, and tries to establish secure connections with victim B as a master and as a slave by using our `BIAS` toolkit. We perform four BIAS attacks:

1) LSC MI: Legacy Secure Connections (LSC) Master Impersonation
2) LSC SI: Legacy Secure Connections Slave Impersonation
3) SC MI: Secure Connections (SC) Master Impersonation
4) SC SI: Secure Connections Slave Impersonation

In the following two paragraphs we describe how we test that victim B is vulnerable to our four attacks.

*a) Master Impersonation:* The attack device impersonates victim A that is not required to be present. We start a secure connection establishment from the attack device to victim B. If victim B does not ask the attack device to authenticate (as in Figure 2), then victim B is vulnerable to LSC MI. If victim B supports Secure Connections, then it is also vulnerable to SC MI, because the authentication procedure is downgraded from secure to legacy (as in Figure 4).

*b) Slave Impersonation:* The attack device impersonates victim A that is not required to be present. We start a secure connection establishment from victim B to victim A. If the attack board switches role from slave to master before accepting the connection request, performs unilateral legacy authentication, and starts the session key negotiation without being asked by victim B to authenticate (as in Figure 3), then victim B is vulnerable to LSC SI. If victim B supports Secure Connections, then it is also vulnerable to SC SI, because the authentication procedure is downgraded from secure to legacy (as in Figure 4).

Our evaluation setup allows to test the BIAS attacks against a target victim in a matter of minutes, and is low cost as it uses cheap hardware and open source software. Our attack device consists of a CYW920819EVB-02 board connected to a Linux laptop. The board costs around 50 USD and any Linux laptop, or even a Raspberry PI, can be used to control the board. Other researchers interested in the BIAS attacks can easily reproduce our setup to test more devices.

### B. Evaluation Results

Table III shows our evaluation results. The first column contains the Bluetooth chip name, and the second column contains the names of the device(s) that we evaluate using such chip. The third and fourth columns evaluate the LSC MI and LSC SI BIAS attacks. The fifth and sixth columns evaluate the SC MI and SC SI BIAS attacks. A solid circle (●) indicates that a chip and the related devices are vulnerable to an attack, and an empty circle (○) indicates that a chip and related devices are not vulnerable. Secure Connections is optional in the Bluetooth standard, and we use - in the SC columns when a chip/device does not support Secure Connections.

*Table III confirms that all the 31 Bluetooth devices (28 unique Bluetooth chips) that we evaluate are vulnerable to our BIAS attacks.* Our list of vulnerable device includes Bluetooth chips, from Intel, Qualcomm (Snapdragon), Cypress (including Broadcom wireless IoT business [11]), Apple, Samsung (Exynos), and CSR (Cambridge Silicon Radio). Furthermore, the list of vulnerable devices includes a mix of proprietary and open source Bluetooth host stacks from Android (Bluedroid and Fluoride), Apple (iOS, iPadOS, and macOS), Linux (BlueZ), Microsoft (Windows 10 and Windows Phone), Cypress, and CSR. Overall, we attack 16 Legacy Secure Connections devices and 15 Secure Connections devices, supporting Bluetooth versions 5.0, 4.2, 4.1, and lower or equal to 4.0.

The only exception is the ThinkPad 41U5005 mouse. The mouse is not vulnerable to our LSC SI attack. In particular, when we let the mouse establish a secure connection with our attack device, even if the attack device switches role and completes the unilateral legacy authentication, the mouse always asks the attack board to authenticate before starting the session key negotiation.

| | | LSC | | SC | |
|---|---|---|---|---|---|
| Chip | Device(s) | MI | SI | MI | SI |
| *Bluetooth v5.0* | | | | | |
| Apple 339S00397 | iPhone 8 | ● | ● | ● | ● |
| CYW20819 | CYW920819EVB-02 | ● | ● | ● | ● |
| Intel 9560 | ThinkPad L390 | ● | ● | ● | ● |
| Snapdragon 630 | Nokia 7 | ● | ● | ● | ● |
| Snapdragon 636 | Nokia X6 | ● | ● | ● | ● |
| Snapdragon 835 | Pixel 2 | ● | ● | ● | ● |
| Snapdragon 845 | Pixel 3, OnePlus 6 | ● | ● | ● | ● |
| *Bluetooth v4.2* | | | | | |
| Apple 339S00056 | MacBookPro 2017 | ● | ● | ● | ● |
| Apple 339S00199 | iPhone 7plus | ● | ● | ● | ● |
| Apple 339S00448 | iPad 2018 | ● | ● | ● | ● |
| CSR 11393 | Sennheiser PXC 550 | ● | ● | - | - |
| Exynos 7570 | Galaxy J3 2017 | ● | ● | - | - |
| Intel 7265 | ThinkPad X1 3rd | ● | ● | - | - |
| Intel 8260 | HP ProBook 430 G3 | ● | ● | - | - |
| *Bluetooth v4.1* | | | | | |
| CYW4334 | iPhone 5s | ● | ● | - | - |
| CYW4339 | Nexus 5, iPhone 6 | ● | ● | - | - |
| CYW43438 | RPi 3B+ | ● | ● | ● | ● |
| Snapdragon 210 | LG K4 | ● | ● | ● | ● |
| Snapdragon 410 | Motorola G3, Galaxy J5 | ● | ● | ● | ● |
| *Bluetooth v$\leq$ 4.0* | | | | | |
| BCM20730 | ThinkPad 41U5008 | ● | ○ | - | - |
| BCM4329B1 | iPad MC349LL | ● | ● | - | - |
| CSR 6530 | PLT BB903+ | ● | ● | - | - |
| CSR 8648 | Philips SHB7250 | ● | ● | - | - |
| Exynos 3470 | Galaxy S5 mini | ● | ● | - | - |
| Exynos 3475 | Galaxy J3 2016 | ● | ● | - | - |
| Intel 1280 | Lenovo U430 | ● | ● | - | - |
| Intel 6205 | ThinkPad X230 | ● | ● | - | - |
| Snapdragon 200 | Lumia 530 | ● | ● | - | - |

TABLE III: BIAS evaluation results. For each of the 28 Bluetooth chips tested, the table shows if the chip is vulnerable (●) or not (○) to the Legacy Secure Connections (LSC) Master Impersonation (MI) and Slave Impersonation (SI) attacks. Additionally, the last two columns show our results for the Secure Connections (SC) MI and SI attacks. (-) indicates that a device does not support Secure Connections.

The table confirms that our BIAS attacks are standard compliant, as the attacks are effective regardless the Bluetooth chip, the Bluetooth host stack, the usage of Secure Connections, and the Bluetooth version number. Furthermore, all devices in the market using any of the vulnerable chips in Table III should be vulnerable to our BIAS attacks. Based on our results, we recommend the Bluetooth SIG to fix the standard as soon as possible, and we provide a list of BIAS attacks countermeasures in Section VIII-C.

## VIII. Discussion

In this section we discuss how to combine our BIAS attack with the KNOB attack [4]. We also comment on the BIAS attack root causes and countermeasures.

### A. Combination of BIAS and KNOB Attacks

Our BIAS attacks, and the KNOB attack proposed in [4] are both standard compliant, but they are different as they reach different goals by exploiting different phases of Bluetooth secure connection establishment. Our BIAS attacks target link key authentication, and they allow the attacker to authenticate as master and slave without having to posses the link key. The

KNOB attack targets session key negotiation, and allows the attacker to lower the entropy of the session key (to brute force it). The KNOB attack alone cannot impersonate a Bluetooth device as the attacker does not posses the long term key.

The BIAS and KNOB attacks can be *chained* to impersonate a Bluetooth device, complete authentication without possessing the link key, negotiate a session key with low entropy, establish a secure connection, and brute force the session key. The combination of the two attacks is novel and powerful. For example, the attacker can impersonate the recipient of a sensitive file and recover the plaintext, or impersonate an unlocker and unlock a device by sending encrypted commands.

### *B. BIAS Attacks Root Causes*

The BIAS attacks evaluated in Section VII are enabled by four root causes (RC) that we identity in the Bluetooth standard. The combination of those root causes allows an attacker to perform master and slave impersonation attacks on LSC and SC. In the following we summarize the root causes:

1) *Integrity.* Bluetooth secure connection establishment is not integrity protected, despite the devices already sharing a long term key ($K_L$). The lack of integrity protection allows an attacker to modify the capabilities of the impersonated victim, including Secure Connections support.
2) *Legacy Mutual Authentication.* Bluetooth Legacy Secure Connections does not mandate to use mutually the legacy authentication procedure [6, p. 559]. When the procedure is used unilaterally there is only one verifier, and the attacker can impersonate the verifier and complete the secure connection establishment without having to authenticate to the victim.
3) *Role Switching.* Bluetooth role switch can be performed anytime after baseband paging [6, p. 595]. In an unilateral authentication scheme this is problematic, as the attacker might start the secure connection establishment procedure as the prover and become the verifier to avoid being asked to authenticate.
4) *Secure Connections Downgrade.* Bluetooth does not enforce the usage of Secure Connections between pairing and secure connection establishment. Hence, two devices who paired using Secure Connections can use Legacy Secure Connections to establish subsequent secure connections. The attacker exploits this fact to downgrade a Secure Connections secure connection establishment to Legacy Secure Connections in order to use the vulnerable legacy authentication procedure.

Table IV shows which root cause is needed to launch the BIAS attacks evaluated in Section VII. We use × when a root causes is needed, otherwise we use - (a hyphen). The lack of integrity protection is needed in any case, as the attacker has to modify the capabilities of the impersonated victim to establish secure connections. The lack of mutual authentication of Legacy Secure Connections is also needed in any case, as the attacker exploits it for Legacy Secure Connections and when downgrading Secure Connections. The role switching is needed for slave impersonations, as the attacker when impersonating the slave has to become the master (verifier) before accepting a connection request. The Secure Connections downgrade is needed only when Secure Connections is in use.

| | LSC | | SC | |
|---|---|---|---|---|
| | MI | SI | MI | SI |
| 1) Integrity | × | × | × | × |
| 2) Legacy Mutual Authentication | × | × | × | × |
| 3) Role Switching | - | × | - | × |
| 4) Secure Connections Downgrade | - | - | × | × |

TABLE IV: Mapping between BIAS root causes and attacks. × indicates that a root cause is needed, a hyphen (-) indicates that a root cause is not needed. LSC means Legacy Secure Connections, SC means Secure Connections, MI is Master Impersonation and SI is Slave Impersonation.

### *C. BIAS Attacks Countermeasures*

The BIAS attacks exploit vulnerabilities in the Bluetooth standard and here we propose three countermeasures to address them. Our countermeasures also address the four attack root causes (RC) presented in Section VIII-B. We do not propose countermeasures acting on top of Bluetooth as they are not fixing the vulnerabilities in the standard.

1) *Integrity.* To mitigate the lack of integrity protection during secure connection establishment, the standard should mandate to use the long term key ($K_L$) to protect the secure connection establishment. The long term key is established during pairing and should be always available before establishing a secure connection. This would prevent manipulation of the Bluetooth capabilities and active man-in-the-middle attacks.
2) *Legacy Mutual Authentication and Role Switching.* To mitigate the lack of mandatory mutual authentication for Legacy Secure Connections and the related issues with role switching, the standard should mandate to always use the legacy authentication procedure mutually. This would force the attacker to authenticate the long term key, even if he switches from slave to master before accepting a secure connection request.
3) *Secure Connections Downgrade.* To mitigate the Secure Connections downgrade attack, the standard should enforce that two devices who paired with Secure Connections are always using it for secure connection establishment. Alternatively, the standard might suggest to notify the user in case of a Secure Connections downgrade and the user should decide whether to accept or reject the downgraded secure connection.

We note that the Bluetooth standard includes “Secure Connections Only Mode” to force devices using only Secure Connections mechanisms, such as secure authentication procedure and AES CCM. That mode is still vulnerable to the reflection attack presented in Section V, and breaks backward compatibility with Legacy Secure Connections devices. We note that none of the devices that we tested is using “Secure Connections Only Mode”.

## IX. Related Work

A recent survey about Bluetooth security is provided by NIST [26]. The survey states that for Bluetooth version from 1.0 up to 3.0 “If device A is the authentication initiator to B, encryption setup will begin after that initial authentication. If the encryption setup being successful is good enough to satisfy B, then B may never bother to attempt to authenticate A”. In our opinion, this claim should be restated saying that for all Bluetooth versions—if Legacy Secure Connections is in use—then device A and device B are not mandated to mutually authenticate before encryption setup. The survey mentions the possibility of impersonation attacks only in the context of broadcast encryption, where a single (master) key is used by all devices to protect the communication, assuming that the attacker knows the key. Our BIAS attacks are much more problematic, because they work in any situation without requiring the knowledge of the long term key.

Secure session establishment is one of two important security mechanism provided by Bluetooth. The other one is pairing, which evolved from legacy pairing to Secure Simple Pairing (SSP). Legacy pairing was broken [16], [33], [29], [20]. SSP was vulnerable to man-in-the-middle attacks [15], [14], [31].

Regarding Bluetooth impersonation attacks, in [19] the authors discuss a relay attack on legacy pairing used to impersonate devices in different Bluetooth networks (piconets). Our BIAS attacks are not simple relay attacks, are effective on both legacy pairing and SSP, and allow to impersonate devices in any piconet. In [22] the authors discuss a replay attack targeting unprotected information. Our BIAS attacks are targeting information protected with the long term key and a simple reply attack is not enough to achieve our goals. In [24] impersonation is discussed in the context of a MitM attack on the pairing phase, and reflection attacks are considered as MitM attacks against authentication. Our BIAS attacks do not require a MitM attacker to be conducted. In [12] the authors are not considering impersonation attacks at all.

An attacker might target implementation bugs on specific Bluetooth devices. For example, the BlueBorne attack vector [5] exploits several flows on Android, iOS, Windows, and Linux implementations. As our BIAS attacks are at the architectural level, they are effective regardless the implementation details.

The Bluetooth cryptographic primitives have been extensively analyzed for weaknesses. The $E_0$ stream cipher (used for Legacy Secure Connections encryption) was investigated [13], and is considered relatively weak in [26]. SAFER+ (used for authentication) was analyzed in [18]. AES CCM (used for Secure Connections encryption) was also analyzed [17], [27] and is FIPS compliant. Nevertheless, our BIAS attacks are effective even with perfectly secure cryptographic primitives.

Several attempts were made to build a low-cost and open source Bluetooth sniffer for over the air eavesdropping [30], [1], [25]. Unfortunately, an affordable and reliable solution is still not here. On the other hand, HCI packets can be sniffed by having root access on the sniffed device, and InternalBlue [23] is providing LMP monitoring for a wide range of chips.

## X. Conclusion

In this work we present the BIAS attacks, our attacks allow to impersonate Bluetooth devices by exploiting vulnerabilities in the specification of Bluetooth authentication and secure connection establishment. We found such vulnerabilities by manual inspection of the Bluetooth standard and leveraging our prior work related to Bluetooth security [4], [3], [2].

As a result of a BIAS attack, an attacker completes secure connection establishment while impersonating Bluetooth master and slave devices, without having to know and authenticate the long term key shared between the victims. The BIAS attacks are standard compliant, and are effective against Legacy Secure Connections (using the legacy authentication procedure) and Secure Connections (using the secure authentication procedure).

The BIAS attacks are the first uncovering issues related to Bluetooth's secure connection establishment authentication procedures, adversarial role switches, and Secure Connections downgrades. The BIAS attacks are stealthy, as Bluetooth secure connection establishment does not require user interaction.

The BIAS attacks are at the architectural level of Bluetooth, thus all standard compliant Bluetooth devices are a potential target. We support this claim, by successfully attacking 31 Bluetooth devices (28 unique Bluetooth chips). Our evaluation sample includes 16 Legacy Secure Connections and 15 Secure Connections devices from several hardware and software vendors, using all major Bluetooth versions. We suggest that the Bluetooth specification should be updated to address our BIAS attacks, and we provide a list of root causes with dedicated mitigations to counter the them.

## References

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

[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Low entropy key negotiation attacks on Bluetooth and Bluetooth low energy. https://eprint.iacr.org/2019/933.pdf.

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

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

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

[6] Bluetooth SIG. Bluetooth Core Specification v5.0. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043, Accessed: 2018-10-28, 2016.

[7] Jiska Classen and Matthias Hollick. Inside job: diagnosing bluetooth lower layers using off-the-shelf devices. In *Proceedings of the Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 186–191. ACM, 2019.

[8] Cypress. CYW20819 Ultra Low Power, BLE/BR/EDR Bluetooth 5.0 SoC. https://www.cypress.com/documentation/datasheets/cyw20819-ultra-low-power-blebredr-bluetooth-50-soc, Accessed: 2019-11-16, 2019.

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

[10] Cypress. ModusToolbox$^{TM}$ Software Environment. https://www.cypress.com/products/modustoolbox-software-environment, Accessed: 2019-11-16, 2019.

[11] Newsroom Cypress. Cypress to acquire broadcom's wireless internet of things business. https://www.cypress.com/news/cypress-acquire-broadcom-s-wireless-internet-things-business-0, Accessed: 2019-11-27, 2016.

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

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

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

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

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

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

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

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

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

[21] Express Logic. Threadx real-time operating system. https://rtos.com/solutions/threadx/real-time-operating-system/, Accessed: 2019-11-23, 2019.

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

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

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

[25] Michael Ossmann. Project Ubertooth. https://github.com/greatscottgadgets/ubertooth, Accessed: 2019-10-21, 2019.

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

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

[28] SEGGER. J-Link EDU: Low-cost J-Link for educational purpose. https://www.segger.com/products/debug-probes/j-link/models/j-link-edu/, Accessed: 2019-11-16, 2019.

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

[30] Dominic Spill and Andrea Bittau. BlueSniff: Eve Meets Alice and Bluetooth. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, volume 7, pages 1–10, 2007.

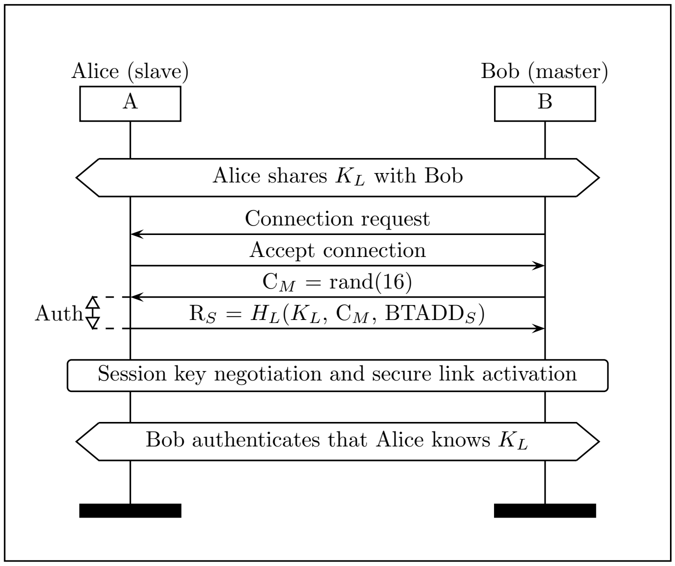
[31] Da-Zhi Sun, Yi Mu, and Willy Susilo. Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard v5. 0 and its countermeasure. *Personal and Ubiquitous Computing*, 22(1):55–67, 2018.

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

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

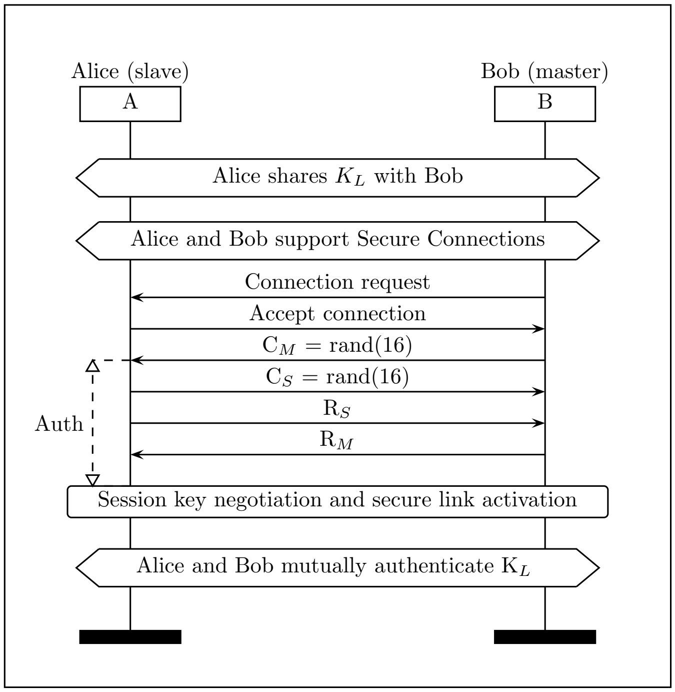
## Appendix

Figure 10 presents the (unilateral) legacy authentication. Figure 11 presents the (mutual) secure authentication.



Alice (slave)
A
Bob (master)
B
Alice shares $K_L$ with Bob
Connection request
Accept connection
$C_M$ = rand(16)
Auth
$R_S = H_L(K_L, C_M, BTADD_S)$
Session key negotiation and secure link activation
Bob authenticates that Alice knows $K_L$

Fig. 10: Legacy authentication procedure. The legacy authentication procedure provides unilateral authentication. Mutual authentication is achieved when the procedure is completed twice, with Bob and Alice as verifier, respectively.



Alice (slave)
A
Bob (master)
B
Alice shares $K_L$ with Bob
Alice and Bob support Secure Connections
Connection request
Accept connection
$C_M$ = rand(16)
$C_S$ = rand(16)
Auth
$R_S$
$R_M$
Session key negotiation and secure link activation
Alice and Bob mutually authenticate $K_L$

Fig. 11: Secure authentication procedure. Alice and Bob exchange $C_M$ and $C_S$ in no particular order. Both compute $R_M$ and $R_S$ using $H_S$. Alice sends $R_S$ and Bob sends $R_M$. If both posses $K_L$, the received value matches the local version.