Finding Traceability Attacks in the Bluetooth Low Energy Specification and Its Implementations

Jianliang Wu[1,2], Patrick Traynor[3], Dongyan Xu[1], Dave (Jing) Tian[1], and Antonio Bianchi[1]

[1]Purdue University, [2] Simon Fraser University, [3] University of Florida

{wu1220, dxu, daveti, antoniob}@purdue.edu, traynor@ufl.edu

Abstract

Bluetooth Low Energy (BLE) provides an efficient and convenient means for connecting a wide range of devices and peripherals. While its designers attempted to make tracking devices difficult through the use of MAC address randomization, a comprehensive analysis of the untraceability for the entire BLE protocol has not previously been conducted. In this paper, we create a formal model for BLE untraceability to reason about additional ways in which the specification allows for user tracking. Our model, implemented using ProVerif, transforms the untraceability problem into a reachability problem, and uncovers four previously unknown issues, namely IRK (Identity Resolving Key) reuse, BD_ADDR (MAC Address of Bluetooth Classic) reuse, CSRK (Connection Signature Resolving Key) reuse, and ID_ADDR (Identity Address) reuse, enabling eight passive or active tracking attacks against BLE. We then build another formal model using Diff-Equivalence (DE) as a comparison to our reachability model. Our evaluation of the two models demonstrates the soundness of our reachability model, whereas the DE model is neither sound nor complete. We further confirm these vulnerabilities in 13 different devices, ranging from embedded systems to laptop computers, with each device having at least 2 of the 4 issues. We finally provide mitigations for both developers and end users. In so doing, we demonstrate that BLE systems remain trackable under several common scenarios.

1 Introduction

Bluetooth Low Energy (BLE) wirelessly connects billions of devices. While BLE makes connecting devices with a range of user interfaces (e.g., from headless systems such as speakers to mobile phones and laptop computers) simple, the protocol's designers understood that such connectivity could make device owners traceable. To avoid this problem, BLE MAC addresses are randomly generated and periodically changed. In practice, however, this defensive measure has been broken repeatedly due to inconsistent adherence to the standard or poor implementations [13, 14, 22, 25, 33].

While MAC address tracking for BLE users has been well studied, a systematic analysis of traceability across the entire protocol has not yet been performed. Accordingly, this paper conducts the first comprehensive analysis of untraceability for BLE. To begin, we define untraceability as the inability of a passive or active adversary to use specification-defined[1], protocol-level unique identifiers to de-anonymize/track a device. From this, we build a formal model by abstracting two operating roles (i.e., central and peripheral[2]), three relations between a tracker and a target device (i.e., the target device has *never* paired, is now *paired*, and was paired but is *unpaired* with the tracker), and five communication scenarios (i.e., pairing, scanning, connection establishment, encrypted communication, and authenticated communication). We then follow each potential identifier as it passes through the protocol to determine whether it represents a traceable attribute. As such, our model covers all connectivity scenarios across the entire lifecycle of BLE communications while removing impossible conditions (i.e., false positives).

In so doing, we make the following contributions:

- **First Comprehensive Study of BLE Untraceability:** We develop a formal model for BLE untraceability that covers all communication modes (e.g., advertising/scanning/encrypted communication, etc.) and all roles. We then implement our model in ProVerif [9] by transforming the untraceability problem into a reachability problem, and identify four design limitations in the specifications, namely IRK reuse, BD_ADDR reuse, CSRK reuse, and ID_ADDR reuse, enabling a total of eight different tracking attacks against BLE. Compared with a diff-equivalence model, our reachability model is sound, as we will discuss in Section 3. Our model is publicly available [3].

- **Confirm Vulnerabilities in Real Devices:** We validate

[1]This paper is based on Bluetooth version 5.3. The findings and mitigations in this paper may also be applicable to other versions.

[2]Central and peripheral correspond to the now deprecated "master" and "slave" roles previously used in the protocol.

our findings using 13 different BLE-enabled devices (ranging from embedded devices and cell phones to laptop computers) and demonstrate that all such implementations contain at least two of these vulnerabilities. Externally, Google rated two of our vulnerabilities as severe, assigned two CVEs, and awarded us a bug bounty.

- **Propose Mitigations:** We have reported our discoveries to the Bluetooth Special Interest Group (SIG), which is responsible for maintaining the standard, and proposed possible updates to the standard. While the SIG considers longer-term fixes, we also discuss several steps that developers and end-users can take to limit their exposure to these vulnerabilities.

A comprehensive understanding of how BLE devices can be tracked is crucial. Especially, the advent of systems like COVID-19 Exposure Notification [28] means that some connectivity scenarios previously considered unlikely (e.g., regularly connecting with devices owned by someone else) are now both practical and can result in a BLE device being tracked. We provide two videos [1, 2] to show how an adversary can track a target device through our newly disclosed vulnerabilities.

2 Background

Bluetooth Classic (BC) uses a fixed Media Access Control (MAC) address, allowing a tracker to trivially track the device by passively sniffing the traffic [18, 45]. Bluetooth Low Energy (BLE) introduces a privacy feature based on the resolvable MAC address mechanism to provide anonymity and prevent naïve device tracking based on a fixed MAC address. The core idea of this feature is that a BLE device uses Resolvable Private MAC Addresses (RPAs) [11, p.2668] generated based on an Identity Resolving Key (IRK). A BLE device must also have an Identity Address (ID_ADDR) [11, p.2666] representing its unique identity. During pairing, the BLE device uses the key distribution to share its IRK and ID_ADDR with a peer device so that the peer device can associate the IRK with the ID_ADDR. By switching to different RPAs, the BLE device's MAC address appears random to a tracker, while the paired peer device can use the IRK associated with the BLE device's ID_ADDR to *resolve* the RPA and recognize the BLE device. Note that if the BLE device enables this privacy feature, it should not include its ID_ADDR in any of its messages [11, p.275] to avoid naïve tracking.

2.1 BLE Privacy Feature

To enable the resolvable MAC address feature, two procedures, RPA generation and resolution, are needed. A target device generates RPAs based on an IRK and uses these RPAs to gain anonymity. Meanwhile, a paired peer device can use the same IRK to resolve the RPAs and identify the target device to guarantee general communication with the target device.

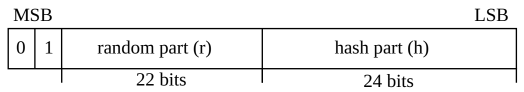

Figure 1: Format of a Resolvable Private Address (RPA). The hash part (h) is generated by a hash function using the random part (r) and a secret key (IRK).

RPA generation. An RPA has two major parts, a random part (`r`) and a hash part (`h`), as shown in Figure 1. To generate a valid RPA, the device first randomly generates `r`. Then the device calculates `h` based on `r` and the IRK using the formula:$h = ah(IRK, 01||r)$, where $ah()$ is a hash function defined in the specification [11, p.1559] based on the AES-128 block cipher. Lastly, a new RPA is generated as `rpa=01||r||h`.

RPA resolution. When the peer device receives an RPA, it extracts `r` and `h` from the RPA. Then, the peer device calculates a `localh` value following the same formula based on `r` and the stored IRK. Lastly, the peer device compares `localh` with `h`. If they match, the peer device resolves the target device's RPA successfully and thus identifies the target device. The device repeats the above steps for each of the IRKs if it has multiple IRKs stored. If `localh` is different from `h` after trying with all stored IRKs, the peer device cannot resolve the RPA and thus cannot identify the target device.

2.2 Key Distribution and Removal

As mentioned, to enable the privacy feature of BLE and use RPAs, a device must have an IRK to generate its RPAs. To ensure a paired peer device can resolve the RPAs and recognize the BLE device, the BLE device shall distribute its IRK to the peer device. The key distribution is performed through the *pairing* procedure, during which two devices establish a mutually trusted relationship. Besides the IRK, the two devices may also generate or exchange other keys, including the *Long-Term Key (LTK)*, which is used for encryption, and the *Connection Signature Resolving Key (CSRK)*, which is used for signing messages. After pairing, the two devices can recognize each other when using RPAs as well as perform encrypted communication using LTK.

For example, consider two devices, Alice and Bob, that have never been paired. If they want to communicate, they pair with each other to establish a trusted relationship. During pairing, Alice stores the keys received from Bob so that she can reuse them (e.g., using the LTK for encryption) when she communicates with Bob again. Similarly, Bob also stores Alice's keys. If Alice does not communicate with Bob anymore, she unpairs from (or forgets) Bob to terminate the trust with Bob. When Alice unpairs from Bob, Alice removes all keys related to Bob. Since the unpairing depends on Alice (i.e., the initiating party) and does not involve any communication with Bob, Bob is unaware of the unpairing and still stores Alice's keys after the unpairing. Nothing prevents a peer device from keeping these keys after unpairing.

2.3 Threat Model

We define an adversary, which we will refer to as a *tracker*, that intends to de-anonymize and track a *target* device using the protocol-level information of BLE.

To be comprehensive, we consider a target BLE device supporting both the central role (e.g., smartphones) and the peripheral role (e.g., smart locks [17]). Additionally, we assume the target device supports all communication scenarios defined in the specification for each role, such as encrypted communication. The target device can be a BLE-only device (e.g., a smart lock) or a BC/BLE dual-stack device (e.g., a smartphone). For a dual-stack device, we assume it does not initiate BC communications since prior research [18, 45] has shown that a BC device can be tracked. In other words, we assume the BC stack is not in use and is in the non-discoverable mode [11, p.1258] on a BC/BLE dual-stack device. Note that a BC device can still respond to requests containing its MAC address (i.e., paging requests [11, p.257]). Likewise, we assume the target device always uses RPAs and does not use a fixed MAC address to prevent naïve tracking by monitoring messages carrying its MAC address. This configuration is the default setting on modern mobile devices, such as iPhones and Android phones (e.g., Google Pixels).

For trackers, we consider both passive and active trackers. A passive tracker is an observer who can only eavesdrop on the BLE channel, while an active tracker interacts with the target device following the specification. We assume that the tracker can deploy one or more devices to track the physical location of the target device, which is a common setting for tracking attacks [22, 25, 33] and real-world tracking scenarios, such as indoor positioning [36, 37, 40].

3 Research Questions

Systematically studying BLE untraceability at the protocol level requires answering fundamental questions (Q) regarding BLE traceability and exposes unique challenges (C). Solutions to these challenges will pave the way to a better understanding of BLE untraceability and privacy.

Q1. Is a BLE device traceable? Existing research [22, 25, 33] considered a device as traceable if and only if its messages carry a fixed MAC address. As such, it is unclear if a BLE device is still traceable when it changes its MAC addresses by enabling the BLE privacy feature. In fact, besides the tracking based on the MAC address, nothing guarantees that there are no other ways to track a privacy feature-enabled BLE device. For this reason, **(C1) without a clear definition of BLE traceability (or untraceability), it is impossible to determine whether a BLE device is traceable or not.**

Approach: Instead of considering only the MAC address as a device's identifier, we consider a piece of information a privacy-sensitive identifier if it is unique per device. After determining these identifiers, we consider a device trackable not only if a tracker can directly observe messages containing any identifier but also if a tracker can identify messages that are indirectly generated using any identifier. In terms of the tracker, we consider two scenarios where a tracker can track a target device, either passively or actively. We define a BLE device as untraceable if neither a passive tracker nor an active tracker can identify and track this BLE device by linking any messages it sends to any of its identifiers. We will present more details in Section 4.1.

Q2. In which communication scenario can a BLE device be tracked? A BLE device may support several communication scenarios, such as encrypted and authenticated communication. In addition, a BLE device can act as different roles behaving differently (i.e., the central role vs. the peripheral role). Moreover, the specification allows various implementation choices. For instance, a BLE stack can use either a public address or a random MAC address as a device's ID_ADDR. Ideally, a BLE device should be untraceable in all BLE communication scenarios no matter what role it is in. For this reason, **(C2) the complexity of BLE communication imposes challenges for comprehensively and systematically studying BLE untraceability.**

Approach: To address this challenge, we use formal analysis techniques and build a symbolic model by abstracting five communication scenarios covering the whole BLE communication lifecycle. In each communication scenario, we model a target BLE device with two different roles, the central role and the peripheral role. Moreover, we explore all three possible relations between a target device and a tracker. With our communication and relation abstractions, our model comprehensively covers all the scenarios of a target device defined in the specification. We will present more details in Section 4.2.

Q3. How do we formally verify BLE untraceability? Formal untraceability usually refers to the notion of "strong unlinkability" [5], which is defined based on Process Equivalence (PE). However, since PE involves relations between traces rather than one single trace, automatically verifying PE is challenging [7, 8, 26, 31]. To avoid the challenges posed by proving PE, existing work defined untraceability in an alternative way. They abstract untraceability into three conditions that can be represented by reachability properties [7, 30]. This definition improves the PE-based definition in several ways. However, these three conditions are themselves difficult to verify, and they require either restriction to a narrow set of protocols [30] or providing multiple manually-defined lemmas [7] (using a theorem prover such as Tamarin [35]). In order to automate untraceability verification, other work [21, 23, 46] adopted a definition of untraceability by using Diff-Equivalence (DE) properties [24], which are another type of reachability properties. If a protocol's DE properties hold, it means that among different executions of the protocol, the message they exchange is the only difference. Though DE-based properties can be automatically verified, this approach may lead to false attacks [10, p.124] [44, p.129]. In Section 5.2, we will further show that modeling BLE untraceability based on DE properties can lead to both false attacks and missed attacks. All these reasons lead to: **(C3) verifying untraceability based on DE properties may lead to false attacks**.

Approach: In this paper, we consider a protocol as untraceable if a tracker cannot distinguish two instances of the protocol she interacts with. Instead of using DE properties to verify untraceability, we explicitly model the tracker's capabilities (e.g., injecting probing messages) and the Bluetooth protocol's states, and transform the untraceability of a BLE device into a reachability problem that can be proved without false attacks. In this way, we can use existing verification tools (e.g., ProVerif or Tamarin) to automatically verify the untraceability of a BLE device, as a sound technique for finding attacks. Specifically, we first extract five pieces of information from the specification as the device's identifiers. Then, we build tracker modules corresponding to each identifier to check whether a received message can be linked to the corresponding identifier of a target device. Lastly, we model a device as it emits a *receiving event* each time it receives a message and emits a *sending event* each time it sends a message. In the tracker modules, the tracker emits a sending event when sending a probing message and emits a receiving event when receiving a message that can be linked to an identifier. In this way, the untraceability of a device can be represented by the *reachability of corresponding event combinations*.

Note that our definition of untraceability is narrower than the DE-based definition, which is narrower than the PE-based definition that is similar to the three-condition-based definition. In theory, the DE-based, the PE-based, and the three-condition-based definitions can capture the attacks discovered based on our definition. Nonetheless, using a more specific definition, our approach overcomes the challenges of the complexity or inability to automate verification in the three-condition-based and the PE-based definitions and the inclusion of false attacks in the DE-based definition. More details can be found in Section 4.3.

4 Model Design and Implementation

In this section, we present our definition of BLE untraceability at the protocol level followed by the design and implementation of our model to verify the BLE untraceability. We implement our model using ProVerif [9] due to its modeling features, such as the *event* feature (which we will introduce in detail in Section 4.3.3). We believe other symbolic reasoning tools, such as Tamarin [35], are also suitable here.

4.1 Definition of BLE Untraceability

We consider BLE untraceability regarding all protocol-level information that can be used as a device's unique identifiers. Furthermore, we call the messages that can be linked to identifiers *linked messages* (LKDmsgs) and others *irrelevant messages* (IRRmsgs). A tracker can link a message to an identifier if the message *contains* this identifier or the tracker can *derive* the identifier from the message. For example, one BLE device may use RPAs for advertising. The advertisement is a LKDmsg of the corresponding IRK since the RPA contained in the advertisement is derived from this IRK.

We consider both passive and active trackers and summarize the following two tracking scenarios:

T1: (Traceability in Passive Tracking) When the target device is communicating with another BLE device, a passive tracker can observe LKDmsgs.

T2: (Traceability in Active Tracking) When the target device is not communicating with another BLE device: 1) an active tracker probes the target device with LKDmsgs and receives any replying messages; 2) an active tracker probes the target device with IRRmsgs and receives LKDmsgs.

Accordingly, we define a BLE device as *untraceable*, if **neither a passive tracker nor an active tracker can identify and track this BLE device by linking any messages to any of the device's identifiers (both T1 and T2 are false)**.

We use K, B, Δ, and A to represent all communication scenarios, roles of the target device, the relations between the target device and the tracker, and the tracker's capabilities, respectively. One BLE communication scenario of the target device θ can be represented by a three-tuple $\langle\kappa,\beta,\delta\rangle$, where $\kappa \in K, \beta \in B, \delta \in \Delta$. We use $untrace\_passive(\theta,\alpha)$ and $untrace\_active(\theta,\alpha)$, where $\alpha \in A$, to denote that the target device is untraceable when it performs communication θ with the tracker α in the passive tracker and active tracker scenarios, respectively. Therefore, whether a BLE device is untraceable can be represented with the following formula:

$$\begin{aligned} \forall\kappa \in K, \forall\beta \in B, \forall\delta \in \Delta, \forall\alpha \in A, \\ untrace\_passive(\langle\kappa,\beta,\delta\rangle,\alpha) \wedge untrace\_active(\langle\kappa,\beta,\delta\rangle,\alpha) \end{aligned} \quad (1)$$

4.2 BLE Communication Abstraction

According to the specification, we abstract BLE communication into five scenarios, namely *pairing* (PAIR), *scanning* (SCAN), *connection establishment* (CE), *encrypted communication* (ENCC), and *authenticated communication* (AUTHC) covering all BLE communication scenarios, as shown in Figure 2. If the BLE device is BC/BLE dual stack, we also add BC *paging* as an extra scenario, since a BC device can still perform paging, as explained in Section 2.3.

In addition, we cover all possible relations between two devices during the entire lifecycle of their BLE communication. Specifically, we assume the target device may have three relations with a tracker: 1. The target device has never paired with

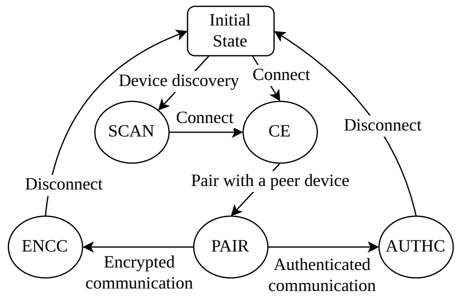

Figure 2: Transitions between the five communication scenarios in the lifecycle of a BLE communication.

a tracker (*never*); 2. The target device is currently paired with the tracker (*paired*); 3. The target device was paired with the tracker but is unpaired now (*unpaired*). Our implementation can be found in our code repository [3].

Pairing (PAIR). Since the security of the pairing procedure is not the focus of this paper and has been explored in prior work [48], we use the out-of-band (OOB) association model in our model and assume that the OOB channel is secure.

In the last step of pairing, the target device shares its IRKs, CSRKs, and ID_ADDRs with a peer device. The LTK is derived by the two devices individually, and thus not shared. As we will discuss in Section 4.3.1, it is not clear in the specification whether a device should use a per-pairing or per-device CSRK. In our model, we consider both cases. In the per-device design, the target device uses one randomly generated CSRK for all pairings, while the target device randomly generates a CSRK for each pairing in the per-pairing design. We also model both the per-device and the per-pairing designs for IRK.

The target device and the peer device are implemented as *process macros* in ProVerif communicating through an open channel. The LTKs, IRKs, CSRKs, and ID_ADDRs are stored in corresponding *tables* (databases for storing data in ProVerif) after the key exchange so that they can be reused in other communication scenarios.

Scanning (SCAN). The scanning starts with a peripheral device broadcasting advertising messages containing its MAC address. Upon receiving the advertising message, a central device sends a scan request message containing the central device's MAC address. Finally, the peripheral device replies with a scan response. All the target device's MAC addresses in the messages are RPAs, as discussed in Section 2.3.

The target device and peer device are also implemented as *process macros*. The target device retrieves its IRK from the IRK *table* to generate RPAs. Since the messages are sent over the air, all messages are exchanged via an open channel.

Connection Establishment (CE). Connection establishment also starts with a peripheral device advertising. Once receiving the advertising message, a central device sends a connection request message containing its MAC address to a peripheral device to establish a connection. Like scanning, the target device also uses RPAs in its messages and retrieves its IRK from the IRK *table* to generate RPAs. The two devices are also implemented as *process macros* and exchange messages via an open channel.

Encrypted Communication (ENCC). After the establishment of a connection, if the upper-layer application requires encryption to guarantee message confidentiality and authenticity, two devices can start encrypted communication. The two devices first derive a session key and a session nonce from two exchanged nonces and the LTK. Then, they encrypt messages using the session key and session nonce and transmit. To generalize the message transmission, we model the central device sending a request message and the peripheral device replying with a response message. The two devices are implemented as *process macros*, and they obtain their LTKs from corresponding LTK *tables* where the LTKs are stored. All nonces and messages are exchanged over the air, and thus modeled as exchanged via an open channel.

Authenticated Communication (AUTHC). After connection establishment, besides ENCC, two devices can also perform authenticated communication to provide authenticity guarantees (without confidentiality). The central device retrieves its CSRK from the CSRK *table* and generates a signature of its message. Then, the central device appends the signature to the message and sends it to the peripheral device. Finally, the peripheral device retrieves the central device's CSRK from the corresponding *table* and verifies the signature. Both devices are implemented as *process macros*, and the authenticated message is sent via an open channel.

Paging (PAGE, BC/BLE only). In the non-discoverable mode, a BC/BLE dual-stack target device does not respond to BC inquiries, but it still responds to BC page requests [11, p.1258]. If the target device receives a page request that contains its BD_ADDR, the target device replies with a page response. Paging on the target device is also implemented as *process macros*. Since the page request and page response are sent over the air, we model that these messages are exchanged via an open channel.

4.3 Verification of BLE Untraceability

To verify BLE untraceability, we first extract all protocol-level information that can be used as a device's unique identifiers (Section 4.3.1). Then, we transform the untraceability of a BLE device into reachability properties by modeling a tracker's capabilities (Section 4.3.2) and representing untraceability as a reachability problem (Section 4.3.3). In this way, we can verify BLE untraceability using existing tools without worrying about false attacks, as we will explain in Section 5.2.

4.3.1 BLE Protocol-Level Identifiers

We systematically study the specification by examining all protocols that must be implemented by a BLE device (i.e., the BLE link layer [11, p.2653], the attribute protocol [11,

Table 1: Protocol-level identifiers of a BLE device and their corresponding major properties. Per-device: a device uses the same value in different pairings. Per-pairing: a device uses different values in different pairings.

Name	Value	Spec. Rec.	Type	Usage
PK	Public Key. Randomly generated	Per-pairing	Short-term	Calculate Diffie-Hellman key
LTK	Long-Term Key. Derived from exchanged values of the two pairing BLE devices, e.g., MAC addresses and nonces	Per-pairing	Short-term	Derive a session key that is used for encryption
IRK	Identity Resolving Key. Randomly generated or assigned during manufacturing [11, p.1588]	Per-device	Long-term	Generate and resolve RPAs
CSRK	Connection Signature Resolving Key. Randomly generated or assigned during manufacturing [11, p.1588]	Per-pairing/ Per-device	Short-term/ Long-term	Generate and verify signatures for authenticated communication
ID_ADDR	Identity Address. Public address or static random address	Per-device	Long-term	Device identity

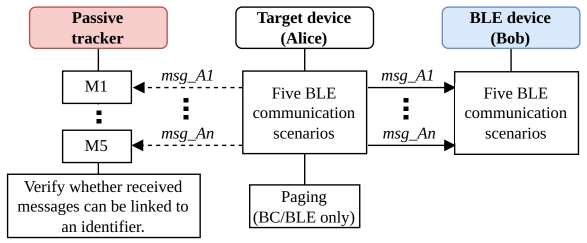
Passive tracker
Target device (Alice)
BLE device (Bob)
M1
M5
msg_A1
msg_An
msg_A1
msg_An
Five BLE communication scenarios
Five BLE communication scenarios
Verify whether received messages can be linked to an identifier.
Paging (BC/BLE only)

Figure 3: Illustration of the five passive tracker modules (M1-M5) in our model.

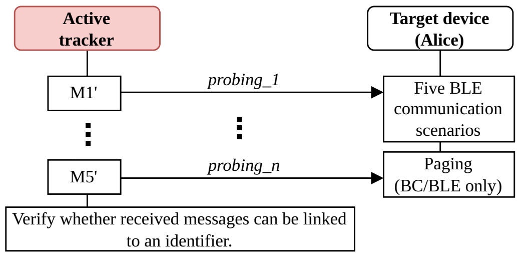
Active tracker
Target device (Alice)
M1'
M5'
probing_1
probing_n
Five BLE communication scenarios
Paging (BC/BLE only)
Verify whether received messages can be linked to an identifier.

Figure 4: Illustration of the five active tracker modules (M1'-M5') in our model.

p.1406], the generic attribute profile [11, p.1464], and the security manager protocol [11, p.1549]). Based on this study, we extract five pieces of information that can be an identifier of a BLE device due to their uniqueness, as listed in Table 1. We categorize the identifiers into two types, i.e., long-term and short-term. The long-term identifiers are permanent or valid until a device revokes them, while the short-term ones are one-time identifiers or identifiers valid per-pairing. Note that all these identifiers will be sent to a paired peer device during pairing.

As the table shows, some identifiers are supposed to be unique per-pairing while others are per-device. Particularly, the specification recommends a BLE device changing its Public Key (PK) after every pairing (per-pairing) [11, p.974]. Since the nonces used to derive the Long-Term Key (LTK) are different in each pairing, the LTK should also change in each pairing (per-pairing). Unlike the PK and LTK, the specification [11, p.1622] states that, "The distributed IRK shall be the same for all devices it is distributed to", which indicates that a device should distribute the same IRK to all devices it pairs with (per-device). The Connection Signature Resolving Key (CSRK) is used to support sending authenticated data over an unencrypted connection [11, p.274]. It is not clear in the specification whether the CSRK should be per-pairing or per-device. We interpret this ambiguity in the specification as both are allowed. Finally, the ID_ADDR, which represents a device's identity, can be either a public address [11, p.2666] or a static random address [11, p.2666]. In a BC/BLE dual-stack device, the public address is also the MAC address of BC (BD_ADDR) [11, p.2122].

4.3.2 Tracker Modeling

Figure 3 and Figure 4 illustrate the overall design of the tracker. We design the tracker following the generic passive and active trackers modeling and offering different tracker instantiations to be plugged in. Specifically, we instantiate one tracker module for each identifier (M1-M5 and M1'-M5' in Figure 3 and Figure 4 respectively). Each tracker module follows the usage of the corresponding identifier according to the BLE specification when checking whether a received message is a LKDmsg of the identifier. We design five tracker modules corresponding to the five identifiers. Each module comprehensively covers all specification-defined ways to check whether a message is a LKDmsg of an identifier. These five modules are adjusted accordingly in the passive and active tracker scenarios and when the target device acts as the central role and the peripheral role, resulting in 20 (5x2x2) tracker modules in total. We now present the detailed design of each of the tracker modules. Our implementation can be found in our code repository [3].

PK tracker module. According to the specification [11, p.1630], a device's PK is shared in plaintext and only used to calculate the Diffie-Hellman key during pairing. Therefore, this tracker module checks whether an eavesdropped or received message is a LKDmsg of the target device's PK by checking whether they are equal.

LTK tracker module. The only use of an LTK is to derive a session key for encrypting the data transmitted in ENCC. Accordingly, the LTK tracker module verifies whether a received message is a LKDmsg of the target device's LTK by trying to derive a session key from the LTK and to decrypt this message using the session key. A message is a LKDmsg of the LTK if this message can be successfully decrypted with the derived session key.

IRK tracker module. The specification uses the IRK to generate RPAs and allows for a distribution of the encrypted IRK to a peer device during pairing. Accordingly, the IRK tracker module leverages two ways to verify whether a received message is a LKDmsg of the target device's IRK. First, for messages that contain RPAs, this tracker module tries to resolve the RPA with the target device's IRK. The message is a LKDmsg of the IRK if the RPA can be resolved. Second, for encrypted messages, our module tries to decrypt the message and check whether the decrypted message is equal to the IRK.

CSRK tracker module. Like IRK, the specification defines two ways to use the CSRK. The first one is to generate a signature of a message during the authenticated communication, and the second one is to distribute the encrypted CSRK to a peer device during pairing. Accordingly, our module verifies the signature of a message (if a signature is present) using the CSRK. The message is a LKDmsg of the CSRK if the signature verification is successful. Our module also tries to decrypt the message and check whether the decrypted message is equal to the CSRK.

ID_ADDR tracker module. According to the specification, to prevent tracking, a target device should not use its ID_ADDR in any of its messages [11, p.2889]. Therefore, the only usage of ID_ADDR by the target device is to send to a peer device during pairing (protected by encryption when sent over the air). Accordingly, the ID_ADDR tracker module tries to decrypt received over-the-air messages and check whether the decrypted message is equal to the ID_ADDR. As discussed in Section 2.3, if the target device is BC/BLE dual-stack, it can still accept BC page requests. As such, this tracker module also constructs and sends a LKDmsg of the ID_ADDR (BC page request) to probe the target device.

4.3.3 Event-Based Untraceability Representation

In ProVerif, an *event* can be used to mark a specific stage of a protocol without affecting its execution. During the verification, events can be *emitted*. If an event is emitted, it means that all operations *before* this event are executed.

To represent untraceability as a reachability property, in the model of Alice (the target device), Alice emits a sending/receiving event after sending/receiving a message. In the passive tracker model, the tracker emits a receiving event if the received message is a LKDmsg of an identifier. Therefore, T1 can be transformed into the following reachability property in ProVerif:

P1: `query msg: bitstring; event(alice_send(msg)) && event(tracker_receive(msg))`

If and only if **P1** is violated (reachable), both events (`alice_send` and `tracker_receive`) happen in a single trace, meaning that the message sent by Alice can be received by the tracker, and that this message is a LKDmsg of one of Alice's identifiers. Thus, the tracker can track Alice. Accordingly, Alice is untraceable by a passive tracker if **P1** holds.

In the active tracker model, the tracker emits a sending event after sending a probing message and a receiving event if receives a replying message. Therefore, T2 can be transformed into the following reachability property in ProVerif:

P2: `query probe: bitstring, rsp: bitstring; event(tracker_send(probe)) && event(alice_receive(probe)) && event(alice_send(rsp)) && event(tracker_receive(rsp))`

Similarly, **P2** can be violated (reachable), if and only if these four events (`tracker_send`, `alice_receive`, `alice_send`, and `tracker_receive`) happen in one single trace. In this trace, Alice receives a probing message from the tracker and replies with a message received by the tracker. Since either the probing message or the replying message is a LKDmsg of one of Alice's identifiers, the tracker can track Alice. Conversely, if **P2** holds, Alice is untraceable by an active tracker.

Note that **P1** and **P2** can be violated in multiple ways since Alice emits an event for *each* message she sends or receives. To find all violations (all the possible ways to track a BLE device), after a violation is revealed, we disable the corresponding events of Alice and run the verification again until no violations are detected. In summary, by combining the tracker modules (M1-M5 and M1'-M5') and the untraceability transformation (**P1** and **P2**), we can automatically verify the untraceability of a BLE device in a systematic and comprehensive manner.

5 Model Evaluation

To discover potential privacy issues leading to device tracking attacks, we use our model to verify the untraceability of Alice (the target device) by querying the two reachability properties (i.e., **P1** and **P2**). We first verify these two properties in the one-session setting, i.e., each process runs once. If a property holds, we will further verify if it still holds in the two-session setting, i.e., each process runs twice. Note that if a property is violated in the one-session setting, it will also be violated in the two-session setting. Therefore, we do not need to re-verify it in the two-session setting. We will discuss the property verification in the settings of three or more sessions in Section 8. Using our model, we were able to identify four privacy issues that allow a tracker to track Alice if she has been paired with a device under the tracker's control. In this section, we present our verification results corresponding to these privacy issues. Lastly, we build an alternative model attempting to verify the untraceability of Alice using DE (diff-equivalence) properties, and we will show the issues of this alternative modeling approach.

5.1 Identified Privacy Violations

When a violation is detected, ProVerif provides a corresponding attack trace. We first analyze these traces and categorize similar traces into the same attack if, in these traces, the tracker follows the same procedure to conduct the tracking attack. In the same attack, Alice might act as different roles in different communication scenarios. Based on the attacks, we summarize four issues (root causes) that enable these tracking attacks.

Among these attacks, some are straightforward (i.e., Attack 1 and Attack 4) because they do not require extra user

Table 2: Detected privacy violations, their corresponding root causes and affected communication scenarios. **1**, **2**, **3**, and **4**: violation caused by Issues 1, 2, 3, and 4, respectively.

#	Alice's Role	Communication	Tracker Type	
			Passive	Active
1	Central	PAIR		1
2				3
3				4
4		SCAN	1	1
5		CE	1	1
6		ENCC		
7		AUTHC	3	3
8		PAGE		2
9	Peripheral	PAIR		1
10				3
11				4
12		SCAN	1	1
13		CE	1	
14		ENCC		
15		AUTHC		
16		PAGE		2

actions nor specialized hardware. Others require additional steps due to the extra prerequisite of the attack, such as user actions in Attacks 2, 3, 6, 7, and 8 and a Software-Defined Radio (SDR) in Attack 5. Among the 8 attacks, Attack 4 is expected when we noticed that BD_ADDR is not used in BLE yet shared during pairing in our earlier BLE experiment. Our analysis further concretizes the violation by generating attack traces. Conversely, all the other 7 attacks are discovered by our formal modeling and analysis without any prior hints. In the rest of this section, we will focus on explaining the detected privacy violations listed in Table 2, the corresponding tracking attacks, and the summarized issues causing these violations. The full result of the verification is shown in Table 7 in Appendix B. All the attack traces can be generated using our code [3].

Issue 1: IRK reuse. The violations indicated by **1** in Table 2 are caused by Alice reusing the IRK for different devices she pairs with. Thus, she distributes the same IRK to all devices (Bob and the tracker) she pairs with. Since the IRK is per-device, the same IRK remains valid even after Alice unpairs from the tracker, and Alice still uses this IRK for future connections. Hence, the tracker can use this IRK to identify and track Alice even after Alice unpaired from the tracker. This issue affects both the central role and the peripheral role because both roles can send messages containing RPAs generated using the IRK. Because IRK is a long-term identifier, the attacks (i.e., Attacks 1, 2, and 3) exploiting this issue are valid until the device explicitly revokes it (e.g., reset network configurations or factory reset).

Violations in the Passive column, rows #4, 5, 12, and 13 share a similar trace, representing the same device tracking attack (**Attack 1**). We will elaborate on this attack and provide a concrete tracking scenario based on this attack trace in Section 6.2.1. In this example, we will demonstrate that a tracker can track an Android phone when its COVID-19 Exposure Notifications system is enabled.

The traces of violations in rows #1 and 9 are similar, leading to the same attack (**Attack 2**). As the trace indicates, after Alice unpairs from the tracker, the active tracker can initiate a new pairing with a random BLE device. The tracker can identify this random BLE device as Alice if the received IRK in the new pairing is Alice's IRK, which is obtained during the previous pairing with Alice.

Violations in the Active column, rows #4, 5, and 9 also share a similar trace, resulting in the same attack (**Attack 3**). The trace shows that when Alice is not communicating with Bob, after Alice unpairs from the tracker, the active tracker can send probing advertising messages to Alice. If Alice replies to the advertising messages with either scan request or connection request messages, the tracker can receive these messages. Since Alice's IRK obtained by the tracker is still valid after Alice unpairs from the tracker, the tracker can use this IRK to resolve the RPA in the message and identify Alice. Compared to Attack 1, Attack 2 and Attack 3 are less practical since they require Alice to perform certain actions, such as replying to an advertising message or accepting a new pairing.

Issue 2: BD_ADDR reuse. The issue leading to the violations denoted by **2** in Table 2 is that Alice reuses her public address, which is also her BC's MAC address (BD_ADDR [11, p.2122]) if Alice is BC/BLE dual-stack, as its ID_ADDR in BLE communications. The traces of these two violations are similar, resulting in the same attack (**Attack 4**). We will detail this attack and provide a concrete attack example in Section 6.2.2, where a tracker can track a BC/BLE dual-stack Android phone as long as its Bluetooth is enabled. Since BD_ADDR is a long-term identifier (e.g., even unchangeable on some devices, as we will further demonstrate in Table 5), Attack 4 remains valid until the device changes its BD_ADDR, or even *forever*.

Issue 3: CSRK reuse. The violations indicated by **3** in Table 2 are caused by the design limitation which allows the CSRK to be long-term (i.e., allowing a BLE device to reuse the CSRK when pairing with different devices). In this case, Alice uses the same CSRK for all devices she pairs with. Like IRK, Alice distributes the same CSRK to all devices she pairs with, and this CSRK remains valid after Alice unpairs from the tracker. As a result, the tracker can use this CSRK to identify Alice by verifying the signature in an authenticated message or directly comparing two CSRKs. This issue only affects the central role since only the central role can send authenticated messages [11, p.1511]. Because CSRK can be a long-term identifier, attacks exploiting this issue (i.e., Attacks 5, 6, and 7) are applicable until the device explicitly revokes its CSRK (e.g., reset network configurations or factory reset).

The violation in the Passive column, row #7 shows that after Alice unpairs from the tracker, the passive tracker can still track Alice when she sends authenticated messages to Bob (**Attack 5**). The attack trace suggests that, when Alice sends authenticated messages to Bob, these messages may be eavesdropped by a passive tracker. Since after Alice unpairs from the tracker, the CSRK stored on the tracker is still valid, the tracker can use this CSRK to verify the signature in the eavesdropped authenticated message. Thus, the tracker can identify Alice if the signature verification is successful. Since Alice sends Bob authenticated messages on general-purpose channels [11, p.2670] with frequency hopping, this attack requires an SDR to sniff the authenticated messages.

The violation in the Active column, row #7 shows that the unpaired active tracker can identify Alice using her CSRK (**Attack 6**). The generated trace indicates that, after unpairing, the active tracker can send probing advertising messages first. If Alice connects to the tracker and sends authenticated messages to the tracker, the tracker can use the same way to verify the signature and identify Alice using her CSRK. Compared to Attack 5, Attack 6 does not require an SDR because Alice directly communicates with the tracker. However, it needs Alice to initiate the connection to the tracker.

The traces detecting violations in rows #2 and 10 are similar, resulting in the same attack (**Attack 7**). After Alice unpairs from the tracker, the active tracker can initiate a new pairing with a BLE device. The tracker can recognize this BLE device as Alice if the received CSRK in the new pairing is the same as Alice's CSRK obtained during the last pairing with Alice. Though Attack 7 does not need an SDR, it requires Alice to approve a new pairing with the tracker.

Issue 4: ID_ADDR reuse. The issue resulting in the violations denoted by 4 in Table 2 is that Alice reuses the ID_ADDR for all devices she pairs with, and this ID_ADDR remains valid after unpairing. These two violations share a similar trace and lead to the same attack (**Attack 8**). As the trace suggests, after Alice unpaired with the tracker, the tracker can proactively initiate a new pairing with random BLE devices and identify a device as Alice if the received ID_ADDR in the new pairing is the same as Alice's. Attack 8 requires Alice to approve a new pairing with the tracker. This issue affects both roles since both roles would distribute the ID_ADDR during pairing. Since ID_ADDR is a long-term identifier (e.g., does not change until resetting network configurations or factory reset), attack 8 is valid until the device explicitly changes its ID_ADDR .

5.2 Comparison with DE-based Model

As discussed in Section 3 and Section 4, verifying BLE untraceability using reachability properties can avoid false attacks identified by existing tools. To better understand the effectiveness of avoiding false attacks, we implement a privacy model of BLE (per-device identifier design) based on DE properties.

In this model, we model the five types of BLE communication and verify the DE properties of the identifiers listed in Table 1. We use the `choice[id,rand_id]` functionality provided by ProVerif to verify DE properties. If the property holds (ProVerif outputs `True`), the tracker cannot tell if the target device uses `id` (the target device's identifier) or `rand_id` (a random identifier) during the communication. It means that the tracker cannot use `id` to track the target device. Since ProVerif verifies a stricter condition than the actual condition when DE properties hold [10, p.62], it may generate false attacks where the stricter condition fails but the DE property holds. We consider an attack as false if and only if the following two conditions are both true: 1) ProVerif outputs `Cannot be proved` but still generates an attack trace, and 2) the attack indicated by the attack trace is not feasible for tracking upon validation. ProVerif may also find a true attack that both the stricter condition and the DE property are violated. In these two cases (i.e., finding false or true attacks), ProVerif outputs `Cannot be proved` because it is not sure whether the DE property is actually violated or not. Note that in these two cases, ProVerif may or may not be able to find an attack trace (no matter false or true attack trace).

Table 3: Results comparison between our reachability-based model and an alternative DE-based model. ●: the alternative model generates false attacks (i.e., false positives). ■: the alternative model cannot generate an attack trace (i.e., false negatives). Cen: central; Per: peripheral; P: passive tracker; A: active tracker

Alice's relation with the tracker, Alice's role, and tracker's type			Communication					
			PAIR	SCAN	CE	ENCC	AUTHC	PAGE
Never	Cen	P	●		●	●	●	
		A	●	●		●	●	
	Per	P	●			●	●	
		A	●			●	●	
Paired	Cen	P	●			●		
		A	●			■		
	Per	P	●		■			
		A	●			■		
Unpaired	Cen	P	●			●		
		A	●			■		
	Per	P	●		■			
		A	●			■		

Table 3 shows the comparison between the verification results generated by our reachability-based model and the alternative DE-based model. As the table shows, compared to our reachability-based model, the DE-based model generates both false attacks and missed attacks. The full verification results of the DE-based model are shown in Table 6 in Appendix B. Internally, DE properties are also represented by reachability properties in ProVerif. However, since our reachability properties implying untraceability are more specific and precise, our model can avoid the false attacks that are inevitable in the DE-based model. Note that, in some cases, false attacks can also be avoided by modeling the operations of a device more precisely, such as the modeling in the three-condition-based definition, as proposed by previous research [7, 30].

For the scenarios denoted by ● in the table, the results of our reachability-based model show that the untraceability holds. However, the results of the DE-based model indicate that the untraceability is violated. After inspecting the corresponding attack traces, we found that these attack traces are false attacks. With further investigation, these false attacks are caused by the stricter condition adopted in proving the DE properties of a protocol with branches. If a protocol contains two branches (i.e., *if* and *else* branches), ProVerif proves the DE property only when the two protocol instances always take the same branch. Thus, ProVerif detects it as a violation if the *if* branch is executed in one protocol instance while the *else* branch is executed in the other instance. However, the protocol still preserves untraceability if the tracker cannot distinguish the *if* branch from the *else* branch.

In the scenarios where the untraceability can be violated (indicated by 🟧), the DE-based model cannot find an attack trace. On the contrary, our reachability-based model can successfully find the correct attack trace. After further study, we believe the failure of attack trace construction of the DE-based model might be caused by other approximations in ProVerif. For example, ProVerif implements the private channel [10, p.123], which is used in the DE-based model, in a way that ignores the order and number of sent messages. The DE-based model and the results can also be found in our code repository [3].

6 Real-World BLE Stack Evaluation

To better understand how the four identified issues affect real-world devices, we perform a comprehensive evaluation covering mobile operating systems (OSes), desktop OSes, embedded OSes, and a Bluetooth stack implementation that is available on a variety of platforms. As shown in Table 4, we select the two most popular mobile OSes (i.e., Android and iOS) [42] and the two most popular open-source embedded OSes that support BLE (i.e., Mbed [6] and Zephyr [19]) [38]. Additionally, we include the three most popular desktop OSes (i.e., Windows, macOS, and Linux) [41] in our evaluation set. Lastly, we evaluate BTstack [27] due to its cross-platform and open-source nature, popularity, and detailed documentation.

We first evaluate whether these 13 devices are affected by the four identified issues in Section 6.1. Then, we present two case studies exploiting IRK reuse and BD_ADDR reuse respectively to demonstrate the practicality of device tracking in the real world in Section 6.2. Lastly, we talk about our reports to related vendors of the devices and their corresponding responses in Section 6.3. In Appendix A, we investigate BLE implementations of tested devices to understand how these devices can change their identifiers to prevent the tracking.

6.1 Evaluation Results

We use one Linux laptop to emulate Bob (i.e., a benign device) to communicate with Alice (i.e., the device to evaluate). We use another Linux laptop to emulate the tracker. We pair Alice with Bob and the tracker, and then unpair Alice from the tracker. Alice is affected by IRK reuse if Bob and the tracker receive the same IRK from Alice. We verify whether Alice is affected by BD_ADDR reuse by comparing the ID_ADDR received by the tracker and Alice's BD_ADDR. Alice is affected if these two MAC addresses are the same. Like IRK reuse, Alice is affected by CSRK reuse if Bob and the tracker receive the same CSRK from Alice. To check whether Alice is affected by ID_ADDR reuse, we compare Alice's ID_ADDR received by Bob and her ID_ADDR received by the tracker. Alice is affected if these two ID_ADDRs are the same.

Table 4 shows the results of our evaluation of the tested devices. As we can see, *all* the tested devices are affected by at least two issues. Note that if a device is affected by one issue, it is vulnerable to all tracking attacks exploiting this issue. For example, if a device is affected by IRK reuse, it is vulnerable to Attack 1, Attack 2, and Attack 3.

During our experiment, we find devices #5, 6, 7, and 11 do not distribute the CSRK during pairing, and thus are not affected by CSRK reuse. Devices #8 and 9 are not affected by BD_ADDR reuse because Zephyr and Mbed do not support the BC stack. Through the experiment and code analysis, we find that Zephyr's BLE stack employs a different IRK and ID_ADDR design. In the default setting, the device is affected by IRK reuse and ID_ADDR reuse. However, it also allows the upper-layer application developers to generate and use several combinations of IRKs and ID_ADDRs at the same time on a BLE device. In this case, whether the device is affected by IRK reuse and ID_ADDR reuse is up to the application developers.

During the evaluation, we also find an implementation issue on Windows 11, caused by not strictly following the specification to implement BLE's privacy feature [11, p.1353]. Specifically, Windows 11 uses its BD_ADDR rather than RPAs as its MAC address when it is the central role. Therefore, a tracker can sniff scan request messages or connection request messages and identify device #10 by simply observing whether its BD_ADDR appears in these messages. In addition, the tracker can also get the BD_ADDR of device #10 by sniffing these two types of messages and track this device by actively probing with page requests using the BD_ADDR.

6.2 Case Study

In this section, we present two practical tracking attacks exploiting IRK reuse and BD_ADDR reuse, respectively. The first case study shows that a tracker can track a target device that enables system services based on BLE advertising, such as the COVID-19 Exposure Notifications system, by launching the Attack 1 attack exploiting IRK reuse. The second case study demonstrates that a tracker can track a BC/BLE dual-stack device even when the target device is *not running any Bluetooth applications* by launching Attack 4 exploiting BD_ADDR reuse.

For these attacks, we assume the tracker can obtain the victim device's identifiers under two scenarios. First, the tracker gets identifiers if the victim device pairs with a device under the tracker's control. Second, the victim device pairs with a benign device which is compromised by the tracker later.

Table 4: Result of whether tested devices are affected by the four discovered issues and the design choice of the identifiers on each tested device. Per-D: per-device design choice. Per-P: per-pairing design choice.

#	Device	Operating System	Stack Implementation	Design Choice			Affected by			
				IRK	CSRK	ID_ADDR	Issue 1	Issue 2	Issue 3	Issue 4
1	OnePlus 8 Pro	Android 13	Gabeldorsche	Per-D	Per-P	Per-D	✗	✗	–	✗
2	Galaxy S21+ 5G	Android 12L	Fluoride	Per-D	Per-P	Per-D	✗	✗	–	✗
3	Pixel XL	Android 10.0	Fluoride	Per-D	Per-P	Per-D	✗	✗	–	✗
4	Pixel 3	Android 9.0	Fluoride	Per-D	Per-P	Per-D	✗	✗	–	✗
5	iPhone 13	iOS 16.2	iOS stack	Per-D	–	Per-D	✗	✗	–	✗
6	iPhone SE (2nd gen)	iOS 14.7.1	iOS stack	Per-D	–	Per-D	✗	✗	–	✗
7	iPhone 8	iOS 13.3.1	iOS stack	Per-D	–	Per-D	✗	✗	–	✗
8	nRF52840 DK	Zephyr 2.7	Zephyr stack	App Dep.	Per-P	App Dep.	✗	–	–	✗
9	nRF52840 DK	Mbed 6.15	Mbed stack	Per-D	Per-D	Per-D	✗	–	✗	✗
10	ThinkPad X1 Yoga	Windows 11	Windows stack	Per-D	Per-P	Per-D	✗	✗	–	✗
11	MacBook Air (M1, 2020)	macOS 13.1	macOS stack	Per-D	–	Per-D	✗	✗	–	✗
12	ThinkPad X1 Yoga	Ubuntu 22.04	BlueZ 5.64	Per-D	Per-P	Per-D	✗	✗	–	✗
13	Dell Latitude 5480	Ubuntu 22.04	BTstack 1.3.1	Per-D	Per-P	Per-D	✗	✗	–	✗

These scenarios are not uncommon in the real world. For example, a user may pair her phone with a BLE smart lock when she moves into an Airbnb apartment and unpair it upon leaving [17]. Similarly, a user may also pair her phone with a rental car and unpair it upon returning the car. In these cases, the smart lock and the rental car could be under the tracker's (e.g., the smart lock or car vendors) control when the user pairs her phone with them. It is also possible that the smart lock and the car are compromised after the user pairs her phone with them. Note that after obtaining the identifiers, the tracker can use other devices, such as Raspberry Pis, to track the phone. For example, the tracker can deploy several devices to perform location tracking or deploy just one device at a specific location (e.g., near the target user's home) to monitor the user's presence.

6.2.1 Attack 1: Tracking a Device with Background Services based on BLE Advertising

Currently, BLE advertising is widely used in the background services of mobile operating systems. Once enabled, these services are running in the background continuously without the user explicitly starting them, such as the COVID-19 Exposure Notifications System [28] on Android and iOS, and the Apple Continuity [4, 14, 43] services. If those devices use RPAs for advertising, they can be affected by IRK reuse.

We take the COVID-19 Exposure Notifications system on Android as a concrete example to show how a tracker can track Alice (e.g., an Android phone). The tracker can take the following steps to identify and track Alice:

1. Alice pairs with the tracker to perform data transmission and sends her IRK (`irkA`) to the tracker during pairing.
2. Alice unpairs from the tracker after data transmission.
3. Alice's COVID-19 service broadcasts advertising messages using RPAs generated from `irkA`.
4. The tracker sniffs the advertising messages and uses `irkA` obtained in step (1) to resolve the RPA in the message.
5. If the RPA resolution is successful, the tracker can identify and track Alice.

We provide a video demo for this tracking attack [2].

This attack is applicable under the real-world Airbnb case mentioned earlier. In this case, the smart lock vendor or the owner of the apartment could be the tracker who obtains the phone's IRK during pairing. Upon leaving, the user unpairs her phone from the lock, however, the IRK obtained by the tracker is still valid. After unpairing, if Alice gets, at any time in the future, physically close to the lock, this lock may receive advertising messages. This is possible, especially when the phone enables background services, such as COVID-19 Exposure Notification. In this case, the lock can resolve the RPAs in the advertising message and identify this phone.

Our experiment shows that the COVID-19 Exposure Notifications system running on devices #2, 3, and 4 is affected by this tracking attack. In addition, the Apple Continuity service running on devices #5, 6, 7, and 11 is also affected. The Exposure Notifications system on iOS (devices #5, 6, 7) is not affected because iOS does not use RPAs for advertising.

6.2.2 Attack 4: Tracking a Bluetooth-Enabled BC/BLE Dual-Stack Device

According to the Bluetooth Special Interest Group (SIG), BC/BLE dual-stack devices, such as smartphones and laptops, account for more than half of all Bluetooth-enabled devices [12]. In this case study, we demonstrate how a tracker can launch Attack 4 to track a dual-stack device *forever*.

We use an Android phone (Alice) as a concrete example to show how a tracker can track this phone when it is *not running any Bluetooth application*. The tracker can follow the following steps to launch the attack:

1. Alice pairs with the tracker to perform data transmission and sends her ID_ADDR, which is also her BD_ADDR, to the tracker.
2. Alice unpairs from the tracker after the data transmission finishes and removes all its identifiers.
3. The active tracker probes Alice with page requests (via `HCI_Create_Connection` commands) using Alice's BD_ADDR obtained in step (1).
4. The tracker can identify and track Alice if she replies with a page request (either the connection can be established,

or the connection is rejected).

We also provide a video demo to demonstrate the practicality of this tracking attack [1].

This tracking attack is also feasible in the Airbnb scenario explained in Section 6.2.1. Compared to the tracking attacks based on IRK reuse, this attack is effective as long as Bluetooth is enabled, and it does not require the target device to run any Bluetooth application, including the background services based on BLE advertising. Our experiment shows that *all* dual-stack devices (devices #1 to 7 and 10 to 13) are affected by this tracking attack.

6.3 Responsible Disclosure

We responsibly disclosed the issues we found to the Bluetooth SIG. The Bluetooth SIG does not consider IRK reuse, BD_ADDR reuse, and CSRK reuse as significant. The reason is that, in order to track a target device, the tracker needs to pair with the target device first, during which explicit user approval is needed. We agree that pairing implies trust, and for this reason, a paired device can trivially track its paired peer devices. However, we believe *a device should not be able to track a previously paired and now unpaired peer device*, as we will further discuss in Section 8. At the time of paper writing, Bluetooth SIG is still investigating ID_ADDR reuse.

We also disclosed our findings to related vendors. Specifically, we reported IRK reuse and BD_ADDR reuse to Google. Google has confirmed our findings, rated both issues as *high severity*, assigned two CVEs (CVE-2021-39673 and CVE-2023-21307), and rewarded us *$10,000 ($5,000 for each issue) as a bug bounty*. At the time of paper writing, Google has fixed IRK reuse with an update for Android 13 by adopting the same strategy as Windows 11, i.e., using a new randomly generated IRK after unpairing from the last paired BLE device [29] (see Appendix A). Though this fix improves the privacy of Android devices, we believe that fundamentally addressing IRK reuse needs specification modifications (as we will explain in Section 7.1). Google has also fixed the BD_ADDR reuse issue, but the fix is not publicly available at the time of paper writing. We also reported these two issues to Apple. Apple considers the tracking based on IRK reuse as matching the Bluetooth standard. In addition, Apple has acknowledged BD_ADDR reuse and is actively working on fixing it. We also reported IRK reuse, BD_ADDR reuse, and the use of BD_ADDR in BLE connection establishment to Microsoft, which has also confirmed our findings. Specifically, Microsoft considers BD_ADDR reuse as a general bug and will fix it in the future. Since Windows does not proactively advertise using RPAs or initiate connections, Microsoft thinks IRK reuse and the use of BD_ADDR do not meet the bar of security vulnerabilities. Moreover, we also reported IRK reuse and BD_ADDR reuse to BlueKitchen (the vendor of BTstack). BlueKitchen has confirmed our findings and considers the tracking based on these two issues as aligned with the current specification. Finally, we also reported IRK reuse and BD_ADDR reuse to BlueZ, Zephyr, and IRK reuse, BD_ADDR reuse, and CSRK reuse to Mbed developers. These vendors are still investigating these issues.

7 Mitigations and Recommendations

To mitigate the four identified privacy issues, we propose updates to the specification. We also provide suggestions to developers and end users on Bluetooth implementation and usage respectively to better protect users' privacy.

7.1 Suggested Specification Updates

To fundamentally address the four revealed privacy issues, updates to the current specification [11] are necessary.

1. Use per-pairing IRKs. The current specification recommends using a per-device IRK, as shown in Table 1, which is the root cause of IRK reuse. To fundamentally address this issue, we propose to update the specification by suggesting that a device should use a fresh random-generated IRK for each pairing to have better privacy protection. In this way, when unpairing from a paired peer device, the target device can revoke the IRK sent to the peer device to prevent tracking attacks based on IRK reuse. Our verification also proves that (rows #32, 33, 40, and 41, columns 7 and 10 in Table 7), the target device cannot be identified or tracked by a peer device after unpairing from it. Therefore, the per-pairing design can significantly improve the privacy of a BLE device.

The per-pairing IRK design works well for the central role but may introduce performance overhead for the peripheral role. The reason is that the central device first scans for advertising messages and then determines which IRK to use. However, the peripheral device has to broadcast advertising messages first, without knowing which central device may connect to it. Therefore, to be recognized by all paired central devices, the peripheral device needs to advertise using RPAs generated by all valid IRKs. Accordingly, it takes longer for two paired devices to reconnect.

2. Use random-generated ID_ADDR. For BC/BLE dual-stack devices, the current specification requires using BD_ADDR as BLE's ID_ADDR [11, p.1615], which is the root cause of BD_ADDR reuse. Therefore, to fundamentally address this issue, a dual-stack device should use a random-generated MAC address (different from its BD_ADDR) as its ID_ADDR for BLE. In this case, when pairing with a tracker, the target device's BD_ADDR will not be leaked to the tracker, preventing tracking attacks based on BD_ADDR reuse, as indicated by our verification in Table 7 (rows #36 and 44, column 10).

Though improving privacy, using random-generated ID_ADDR can affect user experience in the scenario where two dual-stack devices communicate with each other using both BC and BLE. The reason is that these two devices cannot use the Cross-Transport Key Derivation (CTKD) [11, p.1366] feature to derive the link key of BC from the LTK or vice versa because the ID_ADDR is different from the BD_ADDR. As a result, the user needs to pair these two devices twice, once for BC and once for BLE.

3. Use per-pairing CSRKs. The current specification allows using a per-device CSRK, as shown in Table 1, which is the root cause of CSRK reuse. The fundamental mitigation to this issue is using per-pairing CSRKs, as proved in Table 7 (rows #30 and 35, columns #7 and 10). Per-pairing CSRKs is also the implementation choice for most tested devices. As such, we propose to update the specification by disallowing using per-device CSRK and requiring using a fresh random-generated CSRK for each pairing.

4. Use per-pairing ID_ADDR. The current specification suggests using a per-device ID_ADDR, as shown in Table 1, which is the root cause of ID_ADDR reuse. Since the ID_ADDR should not be included in any BLE messages when the privacy feature is enabled [11, p.275], it does not affect the device identification and connection establishment process. As such, we suggest using per-pairing ID_ADDRs to fundamentally address ID_ADDR reuse, as indicated by Table 7 (rows #31 and 39, column 10).

7.2 Suggestions to Developers

We propose the following implementation suggestions to developers to improve BLE privacy while being *compliant with the current specification*.

1. Do not use BD_ADDR in any BLE message. A tracker can sniff a device's BD_ADDR if it is used as the MAC address in a BLE message, such as the use of BD_ADDR on Windows 11 (Section 6.1). As such, a BLE device should use random addresses [11, p.1355] rather than BD_ADDR in BLE communication to prevent leaking a device's BD_ADDR to a tracker when only using BLE.

2. Do not derive IRK from IR. Deriving an IRK from an Identity Root (IR) may result in a fixed IRK, such as the IRK derivation on the nRF52840 DK development board with Zephyr OS (see Appendix A). Therefore, a BLE stack should randomly generate IRKs rather than deriving from the IR.

3. Refresh the IRK after unpairing from the last paired device. As described in Section 6.3, Android 13 refreshes its IRKs when unpairing from the last paired BLE device to mitigate IRK reuse. Windows 11 also adopts the same implementation choice. Though this choice cannot fundamentally address IRK reuse, it can still improve the device's privacy.

7.3 Recommendations to End Users

1. Turn off Bluetooth if not in use. Turning off Bluetooth is an effective way to defend against all tracking attacks based on Bluetooth. For BC/BLE devices, tracking based on BD_ADDR is always effective since a device's BD_ADDR is always valid, especially for unrooted mobile devices. Since devices cannot selectively disable BC while using BLE [34], turning off Bluetooth might be the only way to stop the tracking based on BD_ADDR. Compared with other approaches, turning off Bluetooth is arguably the most practical one for end users.

2. Reset Bluetooth after unpairing. For an Android device, "Reset Wi-Fi, mobile & Bluetooth" revokes its current IRK (Table 5 in Appendix A). Therefore, it is helpful for the device's privacy to perform this resetting after unpairing from a potential tracking device, especially when enabling the COVID-19 Exposure Notifications System. However, resetting Bluetooth on Android also has side effects, such as losing Wi-Fi configurations and information about other paired Bluetooth devices.

3. Unpair from and re-pair with all paired BLE devices after unpairing from a BLE device. For Windows 11 and Android 13 devices, unpairing from all BLE devices will revoke the current IRK (Table 5 in Appendix A). Therefore, after unpairing from one potential tracking device, we suggest unpairing this device from all currently paired BLE devices to refresh the IRK and re-pairing with them. Accordingly, unpairing and re-pairing will mandate extra operations from users.

4. Factory reset after unpairing. For devices running Mbed or Zephyr, factory resetting may remove the stored key database and revoke the current IRK. Similarly, factory resetting an iOS device can also revoke its IRK (Table 5 in Appendix A). Therefore, after unpairing an IoT device, an iOS, or a macOS device from a potential tracking device, factory resetting this IoT device or iOS device can prevent the identification and tracking based on IRK or CSRK. Concretely, factory resetting may be feasible for IoT devices after unpairing, but it might be impractical for other devices.

8 Discussion

Meaning of unpairing. As we discussed in Section 6.3, if a user pairs device A (target device) with device B, it means the user trusts device B. On the contrary, if the user unpairs device A from device B, we claim that the trusted relation is revoked. Thus, after unpairing, device B should not be able to track a previously paired device A. Concretely, there are many scenarios where a user may trust (i.e., pair her device with) a device *temporarily*. For example, in the Airbnb example (Section 6.2.1), the user trusts the smart lock temporarily. A rental car is another example where the user pairs her phone with the car when using it and unpairs her phone from the car when returning it. However, the user currently has no clear way to revoke this trust, since in many cases, the user's device can still be tracked after unpairing. Fundamentally, these issues reveal that the semantics of unpairing is not well-defined in the specification nor implemented coherently across vendors.

Limitations. Our model focuses on BLE untraceability at the protocol level. Thus, it does cover the application level, since it is device-specific, and it is not available at the protocol level. For this reason, device tracking based on the application-level information, e.g., Universally Unique IDentifiers (UUID), has been explored in existing research [25, 33, 50], and is orthogonal to our work. Additionally, our model limits the attacker's capabilities to those modeled in the tracker modules, which are weaker than the capabilities in the Dolev-Yao model. As a result, our model can only discover tracking attacks related to the identifiers covered in the tracker modules. Our model cannot reveal attacks that are not related to the identifiers we extracted from the specification. Nonetheless, since we enumerated all possible usages of the identifiers according to the specification, we believe the modeling of the tracker capabilities related to the identifiers is complete. Our model does not support the setting with three or more sessions. We have tried to verify the properties in the settings of three sessions and an unlimited number of sessions. However, in these two settings, we found that the verification of some properties (e.g., Row 10 Column 8 in Table 7) cannot terminate even after running for 600 hours. Finally, our model does not cover privacy issues caused by side channels, such as timing [16].

9 Related Work

Bluetooth privacy. Fawaz et al. [25] and Kolias et al. [33] studied the privacy information contained in BLE's advertising messages and realized that the fixed field in the advertising message can be used for user tracking. Das et al. [22] found that most fitness trackers utilize fixed MAC addresses for BLE and are vulnerable to user tracking. While none of the existing works systematically studied BLE privacy, our work conducts a systematic study and identifies new privacy issues.

Cominelli et al. [18] and Tucker et al. [45] demonstrated that a BC device can be tracked if BC is in use or in the non-discoverable mode. Ludant et al. [34] showed that the traffic of BC and BLE from the same device can be captured and linked together by a tracker exploiting the traffic transmission timing information. Different from existing attacks [18, 34] that require BC being used to track a device, device tracking based on BD_ADDR reuse is still effective even when the BC is in the non-discoverable mode and not being used.

Formal analysis of Bluetooth. Formal analysis has been applied to the Bluetooth domain, but existing works focused on security properties rather than privacy properties. Several works [15, 20, 39, 48] analyzed the pairing procedure of Bluetooth and found its design weaknesses. Others [47, 48, 49] modeled and analyzed the communications after pairing and revealed new design weaknesses. Our work focuses on the privacy properties of Bluetooth, a field that has not been systematically studied in existing research.

Formal analysis of privacy properties in other protocols. The privacy of other protocols has been studied and analyzed based on different definitions. Arapinis et al. [5] analyzed the Basic Access Control protocol in the French e-Passport based on the PE definition and realized that anyone carrying such a passport can be physically traced. Delaune et al. [23] formally verified three privacy properties that are also based on PE in existing electronic voting protocols. Baelde et al. [7] extended the work of Hirschi et al. [30] and verified the RFID authentication protocol based on the three-condition definition. Dahl et al. [21] analyzed the CMIX protocol [32] using DE and found that it does not preserve privacy in many cases. Wang et al. [46] also used DE to verify a privacy-preserving 5G authentication and key agreement protocol, and proved the protocol resistant to linkability attacks. Different from these, our work proposed a more specific definition of untraceability for BLE, which overcomes the limitations of using aforementioned definitions in automatic verification, as discussed in Section 3.

10 Conclusion

In this paper, we conduct a systematic study of BLE traceability at the protocol level by conducting a formal analysis of BLE untraceability, and we uncover four new privacy issues allowed by the BLE specification. We further evaluate 13 real-world BLE devices and found that *all* of them are affected by at least two of these issues. To mitigate the issues we found, we propose mitigations to address these issues and recommendations to end users.

Acknowledgments

We thank the anonymous reviewers and the shepherd for their valuable comments and suggestions. This work was supported in part by the National Science Foundation (NSF) under Grant CNS-2145744 and the Office of Naval Research (ONR) under Grants N00014-20-1-2128 and N00014-22-1-2671. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of NSF or ONR.

References

[1] Tracking a Bluetooth-enabled Android phone (without any Bluetooth applications running). `https://tinyurl.com/yc4ym757`, 2022.

[2] Tracking an Android phone which enables COVID-19 Exposure Notifications System. `https://tinyurl.com/3d83yu6h`, 2022.

[3] Code repository of our model. `https://github.com/purseclab/btprivacy`, 2023.

[4] Apple. Use Continuity to connect your Mac, iPhone, iPad, iPod touch, and Apple Watch. `https://support.apple.com/en-us/HT204681`, 2022. Accessed: January 10, 2022.

[5] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[6] Arm Limited. Mbed OS. https://os.mbed.com/mbed-os/, 2022. Accessed: January 9, 2022.

[7] David Baelde, Stéphanie Delaune, and Solène Moreau. A Method for Proving Unlinkability of Stateful Protocols. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2020.

[8] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-Aided Cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[9] Bruno Blanchet. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1, 2016.

[10] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. http://tinyurl.com/2d9sv6dm, 2021. Accessed: March 1, 2022.

[11] Bluetooth Special Interest Group. Bluetooth Core Specifications 5.3, 2021.

[12] Bluetooth Special Interest Group. 2022 Bluetooth Market Update. http://tinyurl.com/2xz5eh8s, 2022. Accessed: April 15, 2022.

[13] Guillaume Celosia and Mathieu Cunche. Saving Private Addresses: An Analysis of Privacy Issues in the Bluetooth-Low-Energy Advertising Mechanism. In *Proceedings of the EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, 2019.

[14] Guillaume Celosia and Mathieu Cunche. Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols. *Proceedings on Privacy Enhancing Technologies*, 2020.

[15] Richard Chang and Vitaly Shmatikov. Formal Analysis of Authentication in Bluetooth Device Pairing. *Proceedings of the LICS/ICALP Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA)*, 2007.

[16] Tom Chothia and Vitaliy Smirnov. A Traceability Attack Against e-Passports. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, 2010.

[17] CNET. 9 devices every Airbnb host should put in their rental. http://tinyurl.com/27epk3c9, 2018. Accessed: Novebver 10, 2021.

[18] Marco Cominelli, Francesco Gringoli, Paul Patras, Margus Lind, and Guevara Noubir. Even Black Cats Cannot Stay Hidden in the Dark: Full-band De-anonymization of Bluetooth Classic Devices. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[19] Zephyr Project Community. Zephyr Project. https://www.zephyrproject.org/, 2020. Accessed: February 3, 2020.

[20] Cas Cremers and Dennis Jackson. Prime, Order Please! revisiting Small Subgroup and Invalid Curve Attacks on Protocols Using Diffie-Hellman. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2019.

[21] Morten Dahl, Stéphanie Delaune, and Graham Steel. Formal Analysis of Privacy for Vehicular Mix-Zones. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2010.

[22] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering Privacy Leakage in BLE Network Traffic of Wearable Fitness Trackers. In *Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2016.

[23] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4), 2009.

[24] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic Verification of Privacy Properties in the Applied Pi Calculus. In *Proceedings of the IFIP International Conference on Trust Management*, 2008.

[25] Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. Protecting Privacy of BLE Device Users. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2016.

[26] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking Unlinkability of the ICAO 9303 Standard for e-Passports Using Bisimilarity. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2019.

[27] BlueKitchen GmbH. BlueKitchen BTSTACK. https://bluekitchen-gmbh.com/, 2020. Accessed: February 3, 2020.

[28] Google. Exposure Notifications: Using technology to help public health authorities fight COVID-19. http://tinyurl.com/y6w5k44p, 2020. Accessed: November 1, 2020.

[29] Google. Reconfigure Address policy on last bond removed. http://tinyurl.com/28cbtwzc, 2023.

[30] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A Method for Verifying Privacy-Type Properties: The Unbounded Case. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.

[31] Ross Horne, Sjouke Mauw, and Semen Yurkov. Unlinka-

bility of An Improved Key Agreement Protocol for EMV 2nd Gen Payments. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2022.

[32] Freudiger Julien, Maxim Raya, Márk Félegyházi, Panos Papadimitratos, and Jean-Pierre Hubaux. Mix-Zones for Location Privacy in Vehicular Networks. In *Proceedings of the ACM Workshop on Wireless Networking for Intelligent Transportation Systems (WiN-ITS)*, 2007.

[33] Constantinos Kolias, Lucas Copi, Fengwei Zhang, and Angelos Stavrou. Breaking BLE Beacons For Fun But Mostly Profit. In *Proceedings of the European Workshop on Systems Security (EuroSec)*, 2017.

[34] Norbert Ludant, Tien D. Vo-Huu, Sashank Narain, and Guevara Noubir. Linking Bluetooth LE & Classic and Implications for Privacy-Preserving Bluetooth-Based Protocols. In *Proceeding of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[35] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 2013.

[36] Sugandh Memon, Mehran M Memon, Faisal K Shaikh, and Shakeel Laghari. Smart Indoor Positioning Using BLE Technology. In *Proceedings of the IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2017.

[37] Michael Kwet. In Stores, Secret Surveillance Tracks Your Every Move. `http://tinyurl.com/y4947sg5`, 2019. Accessed: June 30, 2022.

[38] OSRTOS. List of open source real-time operating systems. `https://www.osrtos.com/`, 2023.

[39] Mohit Sethi, Aleksi Peltonen, and Tuomas Aura. Misbinding Attacks on Secure Device Pairing and Bootstrapping. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2019.

[40] Petros Spachos and Konstantinos N Plataniotis. Ble beacons for indoor positioning at an interactive iot-based smart museum. *IEEE Systems Journal*, 14(3), 2020.

[41] Statcounter GlobalStats. Desktop Operating System Market Share Worldwide. `http://tinyurl.com/y26aw45a`, 2023.

[42] Statcounter GlobalStats. Mobile Operating System Market Share Worldwide. `http://tinyurl.com/y2sgeyg3`, 2023.

[43] Milan Stute, Alexander Heinrich, Jannik Lorenz, and Matthias 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 *Proceeding of the USENIX Security Symposium (USENIX Security)*, 2021.

[44] The Tamarin Team. Tamarin-Prover Manual. `https://tinyurl.com/2b3m9tc4`, 2021. Accessed: January 10, 2022.

[45] Tyler Tucker, Hunter Searle, Kevin Butler, and Patrick Traynor. Blue's clues: Practical discovery of non-discoverable bluetooth devices. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[46] Yuchen Wang, Zhenfeng Zhang, and Yongquan Xie. Privacy-Preserving and Standard-Compatible AKA Protocol for 5G. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2021.

[47] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave (Jing) Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, 2020.

[48] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave (Jing) Tian, and Antonio Bianchi. Formal Model-Driven Discovery of Bluetooth Protocol Design Vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[49] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave (Jing) Tian, and Antonio Bianchi. SoK: The Long Journey of Exploiting and Defending the Legacy of King Harald Bluetooth (To appear). In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2024.

[50] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

Appendix

A Identifier Revocation Investigation

As mentioned, the issues in Section 5 stem from the fact that the target device's identifiers received by the tracker are still valid after the target device unpairs from the tracker. Therefore, if the target device revokes (i.e., stops using) such identifiers after unpairing, it can avoid these issues. For this reason, to better evaluate the impact of the tracking attacks based on these issues, we want to figure out *"how often and under what circumstances does a device revoke its shared identifiers?"*

We dynamically test each of the 13 devices listed in Table 4. Specifically, we first pair each of the tested devices with two Linux laptops and then unpair the tested device from the two laptops. During pairing, we capture the traffic on the two laptops, from which we can get the IRK, CSRK, and ID_ADDR

Table 5: IRK, CSRK, and ID_ADDR revocation policies.

#	When the device changes its		
	IRK	**CSRK**	**ID_ADDR**
1	Reset Wi-Fi, mobile & Bluetooth or Factory reset or Unpair from last paired BLE device	Unpair	Never
2	Reset Wi-Fi, mobile & Bluetooth or Factory reset	Unpair	Never
3	Reset Wi-Fi, mobile & Bluetooth or Factory reset	Unpair	Never
4	Reset Wi-Fi, mobile & Bluetooth or Factory reset	Unpair	Never
5	Erase all content and settings	–§	Never
6	Erase all content and settings	–§	Never
7	Erase all content and settings	–§	Never
8	Remove stored key database†	Unpair	Remove stored key database‡
9	Remove stored key database	Remove stored key database	Remove stored key database
10	Unpair from last paired BLE device	Unpair	Never
11	Erase all content and settings	–§	Never
12	Remove stored IRK	Unpair	Never
13	Remove stored key database	Unpair	Never

§: CSRK is not distributed during pairing. †: The default IRK is fixed. ‡: The default ID_ADDR is fixed.

of the tested device. Then, we perform pairing and traffic capturing for a tested device under different scenarios, such as rebooting, resetting settings, factory resetting, and removing the key database. Lastly, we compare the IRKs, CSRKs, and ID_ADDRs in the traffic captured under different scenarios. Besides dynamic testing, we also analyze the source code of the implementations (if available) to confirm our results.

Table 5 shows the results of our experiment. All devices except devices #1 and 10 require performing resetting operations to revoke their IRK, such as "Reset Wi-Fi, mobile & Bluetooth" or factory resetting on Android (devices #2-4). It is noteworthy that, on iOS and macOS (devices #5, 6, 7, and 11), the IRK is revoked only after performing the "Erase All Content and Settings" operation. The IRK remains the same after conducting either the "Reset Network Settings" operation or the "Reset All Settings" operation. For devices #8, 9, 12, and 13, the user needs to explicitly remove the file storing the IRK to revoke it.

The BLE stack on Windows 11 (device #10) uses a different strategy from others to revoke its IRK. The IRK remains the same as long as, at least one BLE device is paired with device #10. After unpairing from the last paired BLE device, device #10 automatically revokes its IRK. Compared with other BLE stack implementations, Windows 11 may still be affected by IRK reuse, but it provides better protection against it. For example, if the user unpairs device #10 from the tracker when no other BLE device is paired, device #10 revokes its IRK after unpairing. Consequently, the tracker cannot identify device #10 using the obtained IRK since it is no longer valid. In a security update [29] for Android 13, Google adopted the same strategy to fix the IRK reuse issue.

The BLE stack on Zephyr has a default IRK calculated from an Identity Root (IR) value defined in the specification [11, p.1624]. Zephyr first tries to read this IR value from the Bluetooth controller. If the controller returns the IR value, Zephyr

Table 6: Verification result of DE-based model with per-device identifiers. ✓: property holds.

#	Relation	Role	ID	Passive Tracker		Active Tracker	
				Result	**Comm.**	**Result**	**Comm.**
1	Never (target device has never paired with the tracker)	Cen	PK	CP1†	PAIR	CP1†	PAIR
2			LTK	CP1†	ENCC	CP1†	ENCC
3			IRK	CP1†	CE	CP1†	SCAN
4			CSRK	CP1†	AUTHC	CP1†	AUTHC
5			ID_ADDR	✓	–	✓	–
6		Per	PK	CP1†	PAIR	CP1†	PAIR
7			LTK	CP1†	ENCC	CP1†	ENCC
8			IRK	✓	–	✓	–
9			CSRK	CP1†	AUTHC	CP1†	AUTHC
10			ID_ADDR	✓	–	✓	–
11	Paired (target device is currently paired with the tracker)	Cen	PK	CP1†	PAIR	CP1†	PAIR
12			LTK	CP1†	ENCC	CP2‡	–
13			IRK	CP3§	CE	CP3§	SCAN
14			CSRK	CP3§	AUTHC	CP3§	AUTHC
15			ID_ADDR	✓	–	CP3§	PAGE
16		Per	PK	CP1†	PAIR	CP1†	PAIR
17			LTK	CP3§	ENCC	CP2‡	–
18			IRK	CP2‡	–	CP3§	CE
19			CSRK	CP3§	AUTHC	CP3§	AUTHC
20			ID_ADDR	✓	–	CP3§	PAGE
21	Unpaired (target device was paired with the tracker, but is unpaired from the tracker)	Cen	PK	CP1†	PAIR	CP1†	PAIR
22			LTK	CP1†	ENCC	CP2‡	–
23			IRK	CP3§	CE	CP3§	SCAN
24			CSRK	CP3§	AUTHC	CP3§	AUTHC
25			ID_ADDR	✓	–	CP3§	PAGE
26		Per	PK	CP1†	PAIR	CP1†	PAIR
27			LTK	CP3§	ENCC	CP2‡	–
28			IRK	CP2‡	–	CP3§	CE
29			CSRK	CP3§	AUTHC	CP3§	AUTHC
30			ID_ADDR	✓	–	CP3§	PAGE

CP1†: ProVerif outputs "Cannot be proved" but generates a *false attack* trace.
CP2‡: ProVerif outputs "Cannot be proved" and *cannot* generate an attack trace.
CP3§: ProVerif outputs "Cannot be proved" but generates a true attack trace.

calculates the default IRK based on this IR value. Otherwise, Zephyr randomly generates the default IRK. In this case, if the IR value returned from the controller is fixed, the default IRK will be fixed. During our experiment, we find that the IR value on the nRF52840 DK development board is fixed, leading to a fixed default IRK. For this reason, the tracker can identify and track the device *forever* if she can get the device's fixed default IRK.

In general, our experiment indicates that different vendors interpret the specification differently. In turn, different BLE stacks implement different strategies to revoke these identifiers offering different privacy guarantees, making it almost impossible for an end user to understand the privacy consequences of using a specific Bluetooth device.

B Full Results of Untraceability Verification

Table 7 lists the full results of the verification using our reachability-based formal model. Note that all the properties that hold in the one-session setting still hold in the two-session setting. Table 6 shows the full results of the verification based on an alternative (diff-equivalence-based) formal model.

Table 7: Verification results of property **P1** and **P2** under all communication scenarios, runtime of the verification under one-session and two-session settings (if needed), and the corresponding identifier if a property is violated. ✗: expected violations. ✗^{1}: violations caused by Issue 1. ✗^{2}: violations caused by Issue 2. ✗^{3}: violations caused by Issue 3. ✗^{4}: violations caused by Issue 4.

#	Relation	Role	Comm.	**Per-device Design†**						**Per-pairing Design§**					
				Passive Tracker			**Active Tracker**			**Passive Tracker**			**Active Tracker**		
				P1	**Runtime** 1Sess/2Sess	**ID**	**P2**	**Runtime** 1Sess/2Sess	**ID**	**P1**	**Runtime** 1Sess/2Sess	**ID**	**P2**	**Runtime** 1Sess/2Sess	**ID**
				1	2	3	4	5	6	7	8	9	10	11	12
1	Never (target device has never paired with the tracker)	Cent.	PAIR	✓	9.4s/1m26s	–	✓	0.4s/1.4s	–	✓	8.4s/1m49s	–	✓	0.1s/0.1s	–
2			SCAN	✓	9.4s/18.1s	–	✓	0.1s/0.1s	–	✓	8.1s/30.1s	–	✓	0.1s/0.1s	–
3			CE												
4			ENCC	✓	9.3s/ 1h19m43s	–	✓	0.1s/0.1s	–	✓	9.0s/ 1h47m27s	–	✓	0.1s/0.1s	–
5			AUTHC	✓	9.0s/18.4s	–	✓	0.1s/0.1s	–	✓	8.2s/22.1s	–	✓	0.1s/0.1s	–
6			PAGE	✓	9.8s/18.1s	–	✓	0.1s/0.1s	–	✓	8.3s/21.9s	–	✓	0.1s/0.1s	–
7		Peri.	PAIR	✓	1.9s/4m54s	–	✓	0.5s/8.8s	–	✓	2.8s/6m29s	–	✓	0.1s/0.1s	–
8			SCAN	✓	1.8s/25.7s	–	✓	0.1s/0.1s	–	✓	2.7s/41.2s	–	✓	0.1s/0.1s	–
9			CE												
10			ENCC	✓	1.9s/ 4h54m56s	–	✓	0.1s	–	✓	2.4s/ 5h40m57s	–	✓	0.1s/0.1s	–
11			AUTHC	✓	1.8s/25.5s	–	✓	0.1s/0.1s	–	✓	2.9s/25.5s	–	✓	0.1s/0.1s	–
12			PAGE	✓	1.8s/25.2s	–	✓	0.1s/0.1s	–	✓	2.7s/25s	–	✓	0.1s/0.1s	–
13	Paired (target device is currently paired with the tracker)	Cent.	PAIR	✓	53.2s/ 22m29s	–	✗	42.2s/–	IRK	✓	37.6s/ 30m10s	–	✓	6.3s/7.2s	–
14							✗	47.1s/–	CSRK						
15							✗	47.4s/–	ID_ADDR						
16			SCAN	✗	40.3s/–	IRK	✗	23.3s/–	IRK	✓	43.3s/35.6s	–	✗	5.2s/–	IRK
17			CE	✗	38.5s/–	IRK	✗	45.6s/–	IRK				✗	5.9s/–	IRK
18			ENCC	✓	59.5s/36.7s	–	✗	4m21s/–	LTK	✓	40.5s/52.9s	–	✗	26.5s/–	LTK
19			AUTHC	✗	39.0s/–	CSRK	✗	49.1s/–	CSRK	✓	43.4s/52.5s	–	✗	6.1s/–	CSRK
20			PAGE	✓	52.5s/34.9s	–	✗	20.7s/–	ID_ADDR	✓	37.1s/52.7s	–	✓	5.9s/7.3s	–
21		Peri.	PAIR	✓	1m36s/ 1h29m6s	–	✗	55.2s/–	IRK	✓	1m57s/ 1h25m33s	–	✓	35.0s/42s	–
22							✗	52.6s/–	CSRK						
23							✗	41.9s/–	ID_ADDR						
24			SCAN	✗	1m38s/–	IRK	✗	23.6s/–	IRK	✗	1m44s/–	IRK	✗	40.9s/–	IRK
25			CE	✗	1m52s/–	IRK	✓	24.3s/22.8s	–	✗	1m10s/–	IRK	✓	25.8s/43.4s	–
26			ENCC	✓	1m33s/ 3m45s	–	✗	7m36s/–	LTK	✓	1m37s/ 2m24s	–	✗	15m25s/–	LTK
27			AUTHC	✓	1m38s/37.6s	–	✓	24.4s/26.5s	–	✓	1m51s/36.9s	–	✓	38.8s/42.5s	–
28			PAGE	✓	2m29s/36.8s	–	✗	26.9s/–	ID_ADDR	✓	1m11s/36.2s	–	✓	35.6s/41.9s	–
29	Unpaired (target device has been paired with the tracker, but is currently unpaired from the tracker)	Cent.	PAIR	✓	37.4s/22m26s	–	✗^{1}	33.9s/–	IRK	✓	42.0s/ 33m38s	–	✓	33.1s/32.4s	–
30							✗^{3}	38.2s/–	CSRK						
31							✗^{4}	43.7s/–	ID_ADDR						
32			SCAN	✗^{1}	41.2s/–	IRK	✗^{1}	22.4s/–	IRK	✓	37.8s/52.3s	–	✓	29.7s/27.9s	–
33			CE	✗^{1}	40.0s/–	IRK	✗^{1}	25.8s/–	IRK						
34			ENCC	✓	48.6s/34.5s	–	✓	23.1s/26.0s	–	✓	39.2s/52.1s	–	✓	30.8s/36.4s	–
35			AUTHC	✗^{3}	42.3s/–	CSRK	✗^{3}	25.0s/–	CSRK	✓	46.0s/52.1s	–	✓	34.0s/36s	–
36			PAGE	✓	47.1s/34.7s	–	✗^{2}	24.6s/–	ID_ADDR	✓	42.3s/51.7s	–	✓	35.1s/36.5s	–
37		Peri.	PAIR	✓	3m1s/ 1h29m6s	–	✗^{1}	43.0s/–	IRK	✓	1m10s/ 1h25m33s	–	✓	33.7s/42s	–
38							✗^{3}	39.0s/–	CSRK						
39							✗^{4}	40.7s/–	ID_ADDR						
40			SCAN	✗^{1}	2m7s/–	IRK	✗^{1}	38.2s/–	IRK	✓	55.2s/35.8s	–	✓	33.9s/41.3s	–
41			CE	✗^{1}	1m57s/–	IRK	✓	25.3/27.9s	–						
42			ENCC	✓	1m45s/ 3m44s	–	✓	7m38s/ 44m8s	–	✓	55.9s/35.5s	–	✓	34.9s/41.4s	–
43			AUTHC	✓	2m55s/37.7s	–	✓	39.7s/39.7s	–	✓	54.2s/35.8s	–	✓	33.6s/41.3s	–
44			PAGE	✓	3m5s/36.9s	–	✗^{2}	32.9s/–	ID_ADDR	✓	34.8s/35.7s	–	✓	35.3s/41.8s	–

† : Alice uses a per-device randomly generated IRK and CSRK, and uses its public address (BD_ADDR) as the ID_ADDR.

§ : Alice uses per-pairing randomly generated IRKs, CSRKs, and ID_ADDRs.