One Tap to Hijack Them All: A Security Analysis of the Google Fast Pair Protocol

Sayon Duttagupta*
COSIC, KU Leuven
Leuven, Belgium
sayon.duttagupta@esat.kuleuven.be

Seppe Wyns*
DistriNet (Group T), KU Leuven
Leuven, Belgium
seppe.wyns@kuleuven.be

Nikola Antonijević
COSIC, KU Leuven
Leuven, Belgium
nikola.antonijevic@esat.kuleuven.be

Dave Singelée
DistriNet (Group T), KU Leuven
Leuven, Belgium
dave.singelee@kuleuven.be

Bart Preneel
COSIC, KU Leuven
Leuven, Belgium
bart.preneel@esat.kuleuven.be

***Abstract*—Google's *Fast Pair Service* (GFPS) extends Bluetooth pairing with one-tap setup and account synchronisation. This paper presents the first comprehensive security analysis of GFPS. By examining 25 commercial accessories from 16 vendors across 17 unique Bluetooth chipsets, we uncover *systemic* enforcement failures of the specification's core security requirements. Moreover, we show that the security failures we have identified in the pairing protocol can be further cascaded, amplifying their impact across the device ecosystem. Although GFPS and Google's Find Hub network are often treated as distinct services within the broader Google ecosystem, we show that failures in one can produce severe consequences in the other. We demonstrate *WhisperPair*, a family of practical attacks that enables unauthorised pairing, silent hijacking of audio devices, and covert account binding that registers a victim's accessory to an attacker's account, thereby enabling persistent location tracking and stalking via Google Find Hub. These vulnerabilities are not isolated incidents but symptoms of *systemic*, ecosystem-wide gaps in implementation, validation, and certification. Our analysis exposes that the source of these flaws lies in GFPS's reliance on fallible, application-layer state checks rather than on cryptographic enforcement, allowing them to propagate across vendors to the end users. To address the root cause, we propose *IntentPair*, a lightweight protocol modification that cryptographically binds the user's pairing intent into the key schedule, eliminating the vulnerability by design. Our findings show how a small usability "add-on" can introduce large-scale security and privacy risks for hundreds of millions of users.**

1. Introduction

Bluetooth accessories have become pervasive in everyday life. Beyond audio headsets and earbuds, the ecosystem now includes smart speakers, fitness trackers, watches, and personal location trackers. These devices enhance convenience by offering untethered access to communication, entertainment, and monitoring services. Yet they all rely on a common entry point: the pairing procedure. Pairing establishes a trust boundary between the device and the user, determining which host is authorised to access microphones, speakers, or location data. For accessories with constrained interfaces, however, this process is inherently fragile. Many have little more than a single button or LED, offering no practical channel for secure code comparison. The result is a persistent tension between usability and security: designers strive for effortless "one-tap" pairing, but in doing so often weaken the authentication guarantees that pairing is meant to provide.

To bridge this usability gap, Google introduced the Google Fast Pair Service (GFPS) in 2017. Fast Pair promises a seamless experience: proximity-triggered setup, account-based key distribution, and optional device-finding features [1]. It relies on Bluetooth Low Energy (BLE) for device discovery and communication. In the Fast Pair model, the peripheral (*Provider*) exposes a Generic Attribute Profile (GATT) service, and the phone (*Seeker*) initiates pairing. By design, Fast Pair hides Bluetooth complexity behind a familiar interface, and paired devices can be synchronised across a Google account. In principle, this design promises the best of both worlds: fast setup and account portability. However, while the security of the core Bluetooth protocols has been studied extensively [2], [3], [4], [5], and recent work has begun to scrutinise proprietary pairing ecosystems [6], [7] and crowdsourced tracking networks [8], the "add-on" protocols built atop them, such as Fast Pair, remain a critical and largely unexamined blind spot. In particular, Fast Pair relies on a simple but security-critical pairing state predicate: a compliant Provider should accept a new unauthorised host only when the device is explicitly in pairing mode. This predicate forms the primary authorisation boundary of the protocol.

Our findings challenge the GFPS security promise. We show that many Fast Pair-certified* accessories violate the

*These authors contributed equally to this work.

*Certified by Google under their Fast Pair programme.

intended trust boundary: Providers frequently accept unauthorised Key-based Pairing requests while in steady state, bonding with an attacker host without the user ever entering pairing mode or consenting. The consequences are immediate. In a crowded commuter train, for example, a victim wearing earbuds may suddenly find their call interrupted as an attacker hijacks the device. The adversary can inject audio or activate microphones. In another scenario, headphones that were never connected to an Android device can be silently attached to the attacker's Google account. This represents a substantial attack surface, as it extends to users outside the Android ecosystem, such as iPhone users who use Fast Pair-certified accessories. Once attached, the accessory is enrolled in Google's *Find Hub* network, enabling covert tracking until the device is factory reset. This leads to the victim's complete loss of location privacy, as the attacker can persistently track them without their knowledge or consent. These attacks require only commodity hardware, succeed within seconds at realistic distances, and affect multiple vendors and chipsets, without any user interaction or physical access. The core defect is the failure to enforce the pairing state predicate. The downstream effects, such as audio hijack, microphone access, and tracking, follow from this initial violation.

We present the first security analysis of GFPS in the field. Examining a wide variety of devices, we uncover *systemic* failures to enforce the specification's[†] core security requirement: that a device must be in explicit "pairing mode" to accept new hosts. Instead, the protocol's security relies on a fallible application-layer state check, where firmware must verify that the user has intentionally entered pairing mode before accepting a Key-based Pairing request. This pairing state predicate is the central protocol invariant that our analysis, tests, and attacks examine throughout the paper. This check is frequently mis-implemented, allowing flaws to propagate across the ecosystem. We leverage these weaknesses to demonstrate *WhisperPair*, a family of practical attacks where a nearby adversary, using only commodity hardware and without any user interaction, bypasses the state check to perform unauthorised pairing. In a more severe variant, we introduce a *covert account-binding* attack that cascades from a Fast Pair flaw into a separate service, Google's *Find Hub* network. By exploiting the pairing defect, an adversary can silently register a victim's accessory under their own Google account, achieving persistent, long-term location tracking of the victim. Although Fast Pair and Find Hub are designed as isolated features, we show how a weakness in one propagates into the other, creating a broader ecosystem risk. This paper studies the security and privacy consequences of these enforcement failures in GFPS. We argue that these vulnerabilities are not isolated bugs but reflect a broader failure of ecosystem governance and quality control across implementations, validation, and certification. Our analysis of the entire GFPS lifecycle reveals that these flaws stem from reliance on policy-based enforcement rather than cryptographic guarantees. Some vendors correctly implement the intended invariant (that a Provider must verify it is in pairing mode before accepting a new host), but many do not. The inconsistency reveals a gap in both *security implementation* and *security compliance*, and non-compliant devices nevertheless pass validation and reach consumers at scale. This leads us to our central research question:

> *Why do vendor implementations and certification processes systematically fail to enforce security-critical pairing requirements in Google Fast Pair, and what are the resulting security and privacy consequences for hundreds of millions of users?*

[†]The GFPS Specification defines Google's documented Fast Pair protocol behaviour, message formats, cryptographic parameters and interfaces used by certified devices.

Contributions

We provide the first detailed, in-the-wild security analysis of the Google Fast Pair Service and show how weaknesses in a usability extension propagate into severe security and privacy risks across the ecosystem. Our main contributions are:

- **Empirical evidence of systemic enforcement failures.** We measure Fast Pair enforcement across **25** commercial accessories from **16** vendors spanning **17** distinct Bluetooth chipsets from **7** chip-manufacturers. Many Providers accept unauthorised Key-based Pairing requests while in normal operation without explicit user action, violating the intended pairing state predicate and breaking the trust boundary between user and accessory.
- **Demonstration of practical attacks.** We design and implement *WhisperPair*, a unified attack family that exploits these enforcement gaps to enable unauthorised attachment, silent hijacking of active audio devices, microphone access without consent, and covert account binding. The covert bind silently registers a victim's accessory under an attacker's Google account, enabling persistent location tracking through Google's *Find Hub* network. The attacks use commodity hardware, require no user interaction, and complete within seconds in proximity settings.
- **Root cause across design, implementation, validation, and certification.** We analyse specification guidance, vendor implementations, and certification artefacts, and trace the failures to a systemic origin. Fast Pair relies on application-layer state checks rather than cryptographic enforcement, and current validation and certification do not verify the pairing state predicate. As a result, non-compliant devices pass validation and reach the market at scale, allowing weaknesses to persist across vendors and firmware generations, thereby compromising the security and privacy of hundreds of millions of devices.
- **Cryptographic enforcement of user pairing intent.** We design a lightweight protocol hardening, called *IntentPair*, that cryptographically binds user pairing intent directly into the Fast Pair key derivation. The modification ensures that pairing succeeds only when the device is in pairing mode or when the requester proves legitimate ownership, thereby eliminating unauthorised attachment by design.

Responsible Disclosure

We followed established responsible disclosure practices, reporting our findings to Google, that acknowledged the issues, accepted the vulnerabilities and classified them as *critical*, their highest severity category [9]. Further details about our correspondence with Google and their mitigation plan are provided in the Sect. Ethics, at the end.

Artefacts

We provide an artefact bundle at https://github.com/KULeuven-COSIC/WhisperPair containing the test harness and attack demonstration code for reproducibility. The artefact requires only a host (laptop or Raspberry Pi) and the target accessory to run the attacks reported in the paper. The materials are intended for defensive research and authorised testing, and to check whether a device is vulnerable.

2. Problem Statement

Bluetooth fast pairing intends to deliver two outcomes simultaneously: frictionless user experience and preservation of the security properties that users associate with explicit pairing. In the Google Fast Pair Service (GFPS) the Provider, typically an audio accessory, exposes a GATT service that accepts Key-based Pairing writes, and the Seeker, typically a handset, initiates the pairing flow. The public specification requires the Provider to ignore pairing writes from unauthorised Seekers when the device is not in pairing mode [1]. This pairing state predicate is the explicit state boundary that should prevent unsolicited attachment during steady-state operation.

Our work examines the hypothesis that this boundary is not enforced consistently in certified devices and that the ecosystem process intended to detect such violations does not reliably do so. We are interested in the security and privacy consequences for ordinary users at realistic ranges, with commodity equipment, and without cryptanalysis of encrypted host traffic.

2.1. Security Invariants and Security Goals

We formalise three protocol invariants, derived from the GFPS specification and user expectations, that capture the conditions a compliant implementation must uphold.

- **Pairing state predicate.** A Provider accepts Key-based Pairing writes *if and only if* it is in pairing mode, which is a user-visible state entered through an explicit physical or software action.
- **Ownership and account association.** Binding to an account or adding a trusted host occurs *only* following an explicit user action indicating reconfiguration.
- **User-facing signalling.** Any change of trusted host or microphone availability *should* be accompanied by signalling that an attentive user can perceive and act upon within seconds.

We consider the relevant security goals to be *confidentiality* of ambient audio near the wearer, *integrity* and *availability* of the user's audio session, and *location privacy* of the user.

2.2. Research Questions

We state our extended research questions explicitly and structure the evaluation to address them.

RQ1 (Root cause). Why do certified Google Fast Pair accessories diverge from the specification, including acceptance of unauthorised Key-based Pairing outside pairing mode and reusing nonces? Are these violations rooted in ambiguous specification guidance, misplaced reliance on application-layer state enforcement or gaps in compliance testing and ecosystem validation?

RQ2 (Enforcement at scale). How consistent are GFPS Providers in enforcing the pairing state predicate across vendors, form factors, and chipset families?

RQ3 (User impact under realistic adversaries). Given commodity equipment and ordinary environments, which attacker capabilities, ranges and latencies are sufficient, which user-facing cues occur, and which concrete harms result (forced host takeover, microphone availability, covert account binding and Google Find Hub enrolment)?

RQ4 (Validation signals and testability). Which observable behaviours constitute a sound oracle for conformance, and to what extent do current public validation artefacts and certification practices exercise these behaviours? What minimal prover–verifier checks would reliably detect non-enforcement?

RQ5 (Mitigation and deployability). Can we harden GFPS so that success is cryptographically conditioned on pairing intent or ownership proof, while preserving wire compatibility and allowing asymmetric roll-out; and which changes to specifications, vendor firmware, and lab tests are necessary and sufficient?

RQ6 (Generalisability). Which lessons extend beyond GFPS to other convenience-first pairing systems that interleave state predicates, account features, cross-protocol and cross-transport transitions?

2.3. Scope of Evaluation

We evaluate a diverse set of commercial accessories spanning multiple brands, form factors, and chipset families, and we use them to measure takeover latency, reliable range, microphone availability without user consent, user-facing cues, auto-recovery behaviour, and the feasibility of covert account association where supported.

We also test robustness against malformed inputs such as invalid elliptic curve points, which indicate implementation hygiene though not required for our main claim. Devices that enforce GFPS correctly are retained as negative controls rather than excluded, highlighting that some devices meet the specification and helping to identify which design or validation choices cause others to fail.

3. Background

This section introduces Bluetooth Classic (BC), Bluetooth Low Energy (BLE), and the Google Fast Pair Service. BC and BLE serve as the foundation for the GFPS protocol. We also introduce the Find Hub extension, which enables location-tracking capabilities for compatible BLE devices.

3.1. Bluetooth Classic

The Bluetooth 5.4 specification [10], introduced in 2023, defines two modes: Basic Rate (BR) and Low Energy (LE). Enhanced Data Rate (EDR) is an extension to the BR mode that enables higher data rates. The BR/EDR mode is also called *Bluetooth Classic*, and is often used to transmit audio between two devices, making it a popular choice for wireless headphones and speakers.

Before data can be sent using Bluetooth BR/EDR, the devices need to discover each other. Discoverable devices, also called *peripherals*, listen for requests on an *inquiry* channel. *Central* devices broadcast inquiry requests, to which peripherals reply with their Bluetooth address and the parameters required to establish a connection.

After this discovery phase, two devices can establish a shared secret by *pairing* with each other. This shared secret is also called the *Link Key*. While early versions of the Bluetooth specification used insecure mechanisms for establishing the Link Key, modern devices use an Elliptic-Curve Diffie–Hellman (ECDH) exchange [11]. While this prevents attackers from deriving the key by passively eavesdropping, devices may still be susceptible to Man-in-the-Middle (MitM) attacks. These attacks can be mitigated if a secure *association model* is used.

The Bluetooth specification defines four association models, depending on the Input and Output (IO) capabilities of the devices. For example, the *Numerical Comparison* model shows a six-digit number on both devices, and requires the user to confirm the pairing if both numbers match. This requires the devices to display a number, and have the user confirm or deny the pairing. Because the numeric comparison values are derived from the underlying ECDH key exchange, MitM attacks will result in a mismatch between the values generated by the two devices. Alternatively, the *Passkey Entry* model can be used if one of the devices cannot display a number. If an alternative mechanism is available to securely transmit data, the *Out of Band* model can be used. Finally, the *Just Works* model is similar to the Numerical Comparison model, but it does not show a number to the user. While this is the only option for certain devices, it does not protect against MitM attacks.

After establishing a (secure) connection, devices can exchange information as specified by a *profile*. A Bluetooth profile describes an interface that devices can use to interact with each other. For example, the Advanced Audio Distribution Profile (A2DP) specifies how audio can be streamed from one device to another.

3.2. Bluetooth Low Energy

In 2010, the Bluetooth 4.0 specification [12] introduced Bluetooth Low Energy (BLE), aiming to bring Bluetooth to resource constrained environments. Similarly to Bluetooth BR/EDR, a BLE device can implement profiles to exchange information. The Attribute Protocol (ATT) allows a device (the server) to expose a set of attributes to another device (a client). Clients can read or write to an attribute, and they can register for notifications when the attribute value changes. Every attribute has a value, type, permissions, and a handle. The handle is used to uniquely identify the attribute, and is sent when the client wants to perform a read or write operation. The set of permissions can be used to enforce encryption or authentication before an attribute can be read or written. The attribute type is a Universally Unique Identifier (UUID) that specifies what kind of value the attribute represents. ATT does not place constraints on the attribute value, as this should be specified by higher-level profiles like the Generic Attribute Profile (GATT)

GATT builds upon ATT by hierarchically structuring attributes using *profiles*, *services*, and *characteristics*. A GATT profile specifies how data is structured using services. For example, the Blood Pressure Profile specifies how devices should expose blood pressure measurements. It requires two services: the Blood Pressure Service and the Device Information Service. The goal of a service is to provide some functionality of the device using characteristics. A characteristic represents a data value along with some properties that specify how it should be accessed or modified. For example, the Device Information Service contains a Model Number characteristic. This characteristic should contain the model number of the device, and it cannot be written by a client. While some services are standardised by the Bluetooth Special Interest Group (SIG), GATT also enables manufacturers to implement proprietary protocols, such as Google Fast Pair.

3.3. Google Fast Pair Service

The Google Fast Pair Service, introduced in 2017, streamlines Bluetooth pairing to improve user experience. Originally designed for Android, it now extends across the Google ecosystem, including ChromeOS. Without Fast Pair, users must manually select an accessory in Bluetooth settings and, depending on the association model, verify a numerical passkey. Fast Pair automates this process by discovering nearby devices and enabling pairing with minimal interaction. Rather than replacing the BR/EDR pairing process, it acts as an add-on that establishes a secure pairing without user interaction. Fast Pair uses BLE and is implemented as a custom GATT profile. GFPS defines two roles: the *Provider* and the *Seeker*. The Provider acts as a GATT server, typically an accessory such as headphones, while the Seeker, usually a smartphone, operates as the client initiating pairing. Although the specification is public, manufacturers must register their accessories with Google before distributing them commercially.

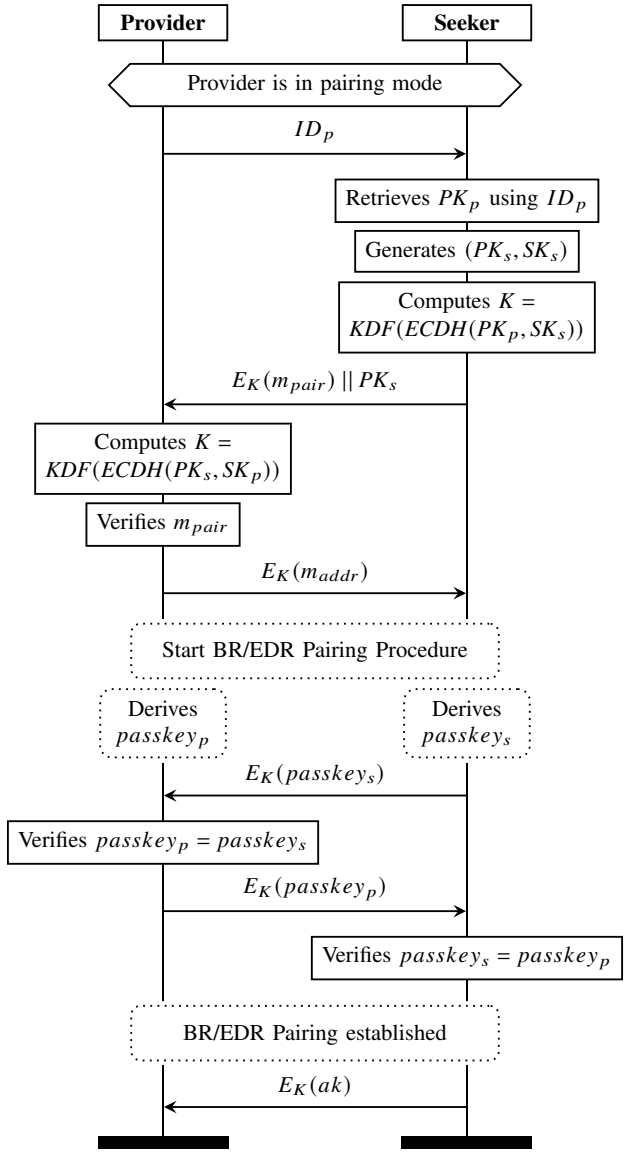


Figure 1. The pairing process of Fast Pair between a Provider and a Seeker. Steps described in the Fast Pair specification are shown in solid lines, while the parallel BR/EDR pairing procedure is denoted using dashed boxes.

3.3.1. Model Registration. Upon registering an accessory with Google, manufacturers receive a 24-bit Model ID and an *Anti-Spoofing* public/private key pair. The key pair is used to derive a shared secret during the Fast Pair procedure, and is static for each model. The private key should be stored securely on the Provider, while Seekers can retrieve the public key of a model by calling a Google API. Before the public key of an accessory is made available on this API, the model needs to be certified by Google. This process involves self-tests using a validator app, and shipping the device to a certified 3rd party lab. The goal of the certification is to ensure that the Fast Pair specification is implemented correctly, before the accessory is made available to consumers.

3.3.2. Initial Pairing. Figure 1 illustrates the initial pairing procedure between a Provider and a Seeker. Messages are exchanged through notifying and writing to GATT characteristics. Each message includes a random nonce to prevent replay attacks.[‡]

When a Provider enters pairing mode, it begins broadcasting its Model ID using the Fast Pair service UUID. Seekers continuously scan for such advertisements and prompt the user to pair when a nearby Provider is detected. A Google API returns metadata for a given Model ID, including the Anti-Spoofing public key. If the user opts to proceed, the Seeker initiates the Fast Pair procedure.

Rather than immediately performing BR/EDR pairing, the Seeker first establishes an unencrypted BLE connection to the Provider. It generates an elliptic curve public and private key pair on secp256r1, and performs an ECDH exchange with the Provider's Anti-Spoofing public key. The shared secret is processed by SHA-256, and the first 128 bits form the *Anti-Spoofing AES key* K. This key is used to encrypt subsequent BLE messages, and we refer to it as the *session key*. The Seeker then writes to the Key-based Pairing (KbP) characteristic, transmitting an encrypted payload $E_K(m_{pair})$ together with the public key PK_s that was used to derive K. The encrypted payload contains a nonce, the Provider's BLE address, and some connection parameters.

Upon receiving a write request that includes a public key on the KbP characteristic, the Provider checks whether it is currently in pairing mode.[§] Then, it derives the same session key K using its private key SK_p and the received PK_s. The Provider decrypts the payload and verifies that it contains its own BLE address. If the check succeeds, the session key is stored for the remainder of the BLE connection. The Provider replies with a message m_{addr} containing a fresh nonce and its BR/EDR address.

Once the Seeker receives and decrypts $E_K(m_{addr})$, it begins BR/EDR pairing in parallel with the already established BLE connection. To ensure the pairing uses Numerical Comparison as the association model, the Seeker sets its IO capabilities accordingly. If the Provider receives a BR/EDR pairing request whose IO capabilities do not support numerical comparison, it must reject the pairing. If the pairing proceeds, the Provider awaits a write to the Passkey characteristic.

As part of the BR/EDR pairing procedure, the Provider and Seeker each derive a passkey based on the exchanged cryptographic keys. The purpose of these passkeys is to prevent MitM attacks, as the presence of an attacker would cause the two parties to derive different passkeys. This passkey derivation process is described in the Bluetooth Core specification [10] and occurs independently of Fast Pair. However, rather than requiring the user to visually

[‡]While the specification incorrectly calls these values *salts*, we refer to them as *nonces*.

[§]This check constitutes the critical authorisation predicate of initial pairing: if it is omitted or mis-implemented, an unauthorised Seeker may be accepted as a legitimate host.

compare the passkeys, Fast Pair automates this verification by exchanging the keys over BLE.

When the BR/EDR pairing process halts to wait for passkey confirmation, the Seeker encrypts and sends its passkey to the Provider. The Provider decrypts this value and compares the received passkey $passkey_s$ with its expected passkey $passkey_p$. The pairing should be confirmed only if these match, but the Provider must always send a reply containing $passkey_p$. The Seeker then compares its expected passkey with the value returned by the Provider and confirms pairing only if the two match.

Once BR/EDR pairing is complete, the Seeker generates a fresh 128-bit *account key* ak and transmits it to the Provider, encrypted under the session key K. The Provider decrypts and stores this long-term key in non-volatile memory, enabling automatic re-pairing by other devices associated with the same user account.

3.3.3. Account Keys and Subsequent Pairing. Account keys allow an accessory to be associated with a user's Google account. When the Provider is active but not in pairing mode, it advertises an *account key filter* instead of its Model ID. The advertisement payload contains a Bloom filter [13] over the hashes of all stored account keys, each combined with a nonce that is regenerated whenever the Provider rotates its BLE address. This nonce appears only in the advertisement and is never reused. It allows Seekers to test, in a privacy-preserving manner, whether any of their locally stored account keys match those held by the Provider. If a match is detected, the Seeker may initiate pairing even though the Provider is not explicitly in pairing mode.

In this subsequent pairing path, the Seeker uses its long-term account key ak directly as the session key rather than generating an ephemeral key pair. Its initial message therefore contains no public key PK_s. Upon receiving such a message, the Provider attempts decryption under each stored account key until one produces a valid plaintext that parses correctly as a raw request and contains the Provider's BLE address. Only that account key is then adopted as the active session key for the connection. After this point, the remaining steps mirror the initial pairing shown in Fig. 1.

The crucial distinction between initial and subsequent pairing is therefore the presence of PK_s in the first message. A public key indicates that an unauthenticated Seeker is attempting to pair, whereas its absence allows the Provider to check whether the requester holds a known account key. Providers should only accept messages from unauthenticated Seekers when they are explicitly in pairing mode. By contrast, subsequent pairing outside pairing mode is intended to be authorised by possession of a valid account key. This mechanism enables trusted devices to connect without forcing the Provider back into pairing mode, a design intended to reduce user friction [14].

3.4. Find Hub Network Extension

The Fast Pair specification defines several optional extensions, including battery reporting and support for the Find Hub network.[¶] Find Hub uses crowdsourced location data from Android devices to track location tags, smartphones, and accessories that implement its extension. This extension introduces a *beacon actions* GATT characteristic. As in the initial pairing process of Sect. 3.3.2, the BLE connection remains unencrypted; instead, each write to the characteristic must carry an authentication tag computed with the MAC algorithm HMAC-SHA256, using a key derived from the user's account key. To ensure that only the legitimate owner can enrol a device into Find Hub, the Provider designates a single *Owner Account Key* (OAK), selected as follows.

When a Provider supporting Find Hub is paired for the first time, the account key written during that pairing is marked as the OAK. The designation is permanent and can only be cleared by factory resetting the Provider. Since all authentication keys for Find Hub provisioning messages are derived from the OAK, only a Seeker holding this key can initialise the accessory as a Find Hub beacon. An OAK is selected only when an accessory is paired with a Fast Pair compatible Seeker.

4. Protocol Security Tests

While Google ensures that Android devices support Fast Pair, manufacturers need to provide their own implementation of a Fast Pair Provider. Google strongly recommends manufacturers to partner with System Integrators that offer SDKs easing the integration of Fast Pair [15]. Since Fast Pair enables device pairing, audio switching, and location tracking, validating the correctness of an implementation is critical. Our methodology began by manually analysing the public Fast Pair specification and identifying a small set of security-critical invariants whose violation could enable unauthorised attachment or weaken trust establishment. We then operationalised these invariants as concrete conformance tests and implemented them in an automated test harness that we executed in a controlled setting. Rather than attempting an exhaustive analysis of all Fast Pair behaviour, this section focuses on three such tests: pairing state predicate enforcement, nonce reuse, and invalid curve handling. These tests are not intended to be exhaustive and do not rule out further protocol- or implementation-level issues.

4.1. Threat Model

Our test suite adopts the same threat model as the Fast Pair specification, which corresponds to the standard Dolev–Yao model [16]. The adversary is a proximal attacker equipped with a commodity smartphone, laptop, or Raspberry Pi that supports BLE and BR/EDR. Such an attacker can operate in public or semi-public environments, including trains, cafés, offices, corridors, and lecture halls. The adversary has no physical access to the victim's devices and no prior pairing with the accessory. However, they can

[¶]Not every successful Fast Pair attachment yields tracking capability. This requires Find Hub support and the corresponding owner account key state.

scan BLE advertisements, establish a GATT connection to any Provider within range, issue writes to the Key-based Pairing characteristic, and initiate a BR/EDR connection whenever the Provider transitions to the corresponding state.

Attack Preconditions. The attacks described in this work require the following conditions to hold: (i) the target device is powered on, (ii) the target device supports the Google Fast Pair protocol, (iii) the attacker is within BLE communication range, (iv) no prior pairing, shared secrets, or trusted relationship exists between the attacker and the target device. These preconditions reflect the intended operating environment of Fast Pair and are consistent with real-world usage scenarios. In addition to these conditions, the state of the target device determines which attacks can be performed:

- **Pairing State Predicate Enforcement.** (Sect. 4.3) This attack requires no user interaction. It assumes the device is not in pairing mode, but this is not a strict requirement as the attack remains viable regardless.
- **Nonce Reuse and Invalid Curve Attacks.** (Sections 4.4 and 4.5) These attacks require the device to be in pairing mode, triggered by a device-specific physical interaction such as pressing a button.
- **Find Hub Network Enrolment.** (Sect. 5.2) Unauthorised enrolment in the Find Hub network requires that the target device has not yet selected an OAK. This explicitly targets devices that have never been paired with an Android device.

4.2. Assumptions and Non-goals

We assume the Provider implements the public cryptographic components of Key-based Pairing at a functional level, including ECDH on secp256r1 and symmetric key derivation as specified in the public documentation. We do not assume access to vendor source code, proprietary certification materials, or private keys. Our tests do not rely on baseband or kernel vulnerabilities, nor on jamming or non-standard transmit power settings. We do not attempt to decrypt legitimate handset traffic or perform any cryptanalysis of it. Our goal is to identify implementation flaws and protocol-level oversights that allow pairing to proceed while still compromising security. For instance, a Provider that fails to encrypt a required payload as required, will simply cause the Seeker to reject the exchange, which we assume would be caught during certification. Such cases fall outside our scope.[‖] The remainder of this section introduces three compliance tests: pairing state predicate enforcement, nonce reuse, and invalid curve handling.

4.3. Pairing State Predicate Enforcement

A normal Fast Pair procedure begins when the Provider advertises its Model ID. Seekers can then open a BLE connection and write to the Key-based Pairing characteristic. This characteristic remains discoverable outside pairing mode, allowing already trusted devices to pair automatically. To prevent unauthorised Seekers from pairing, the Provider should check whether it is in pairing mode upon receiving any write that contains a public key. Incorrect enforcement does not affect legitimate pairing, yet it is central to security.

To validate this behaviour, we consider a Provider P that is already paired to a Seeker S. When not in pairing mode, P advertises an account key filter rather than its Model ID. Although the Model ID is absent from these advertisements, the BLE address remains visible, allowing us to connect and write to the Key-based Pairing characteristic. Deriving a valid session key also requires the Provider's Anti-Spoofing public key.

Every Fast Pair model has a static Anti-Spoofing key pair (Sect. 3.3.1). Android devices obtain the corresponding public key through an undocumented Google API that accepts a Model ID as input. Since Model IDs occupy a 24-bit space, all certified models can be enumerated by querying this API. Although the Provider broadcasts its Model ID only in pairing mode, the same value can be retrieved by reading the Model ID characteristic. In practice, several devices also leak the model name in advertisements, which can reveal the ID.

After recovering the Model ID and public key, we generate an ephemeral secp256r1 key pair, compute the ECDH shared secret, apply SHA-256 to its x-coordinate, and truncate to 128 bits to obtain the Anti-Spoofing AES key. The 16-byte pairing request payload contains the Provider's BLE address, a nonce, and flags, encrypted under AES-128 in ECB mode.[**] The request is formed by concatenating this ciphertext with the ephemeral public key.

We enable notifications on the Key-based Pairing characteristic and send the write request while the Provider is not in pairing mode. Since the request contains a public key, a compliant Provider must ignore it. Any reply indicates that the pairing state predicate is not enforced. Such a reply contains the Provider's BR/EDR address, encrypted under the derived session key. Figure 2 summarises the workflow; subsequent protocol steps follow the normal procedure described in Sect. 3.3.2.

4.4. Nonce Reuse

The Fast Pair specification mandates that each message include a nonce. Providers should validate nonces to reject replays, or alternatively, rotate their BLE address before the next pairing request. Because the address is embedded in the initial message, any request containing an old address should be ignored. Our second test evaluates whether manufacturers prevent replays by validating nonces or rotating addresses. We consider a Provider P in pairing mode that advertises Fast Pair support through its Model ID. As in Sect. 4.3, we

[‖]More generally, our methodology is designed to test a targeted set of protocol-conformance properties and should not be read as ruling out other protocol-level or implementation-level weaknesses beyond those captured by the three tests below.

[**]We note that ECB here is used only for a single fixed-size block within the protocol and therefore does not by itself imply the standard multi-block pattern leakage associated with ECB, and our attacks do not rely on ECB misuse.

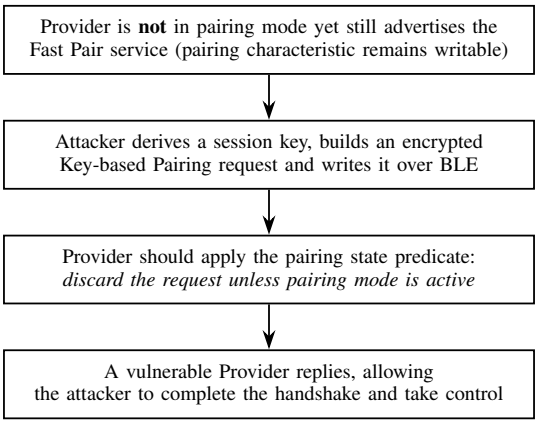


Figure 2. High-level flow illustrating enforcement of the pairing state predicate. A compliant Provider ignores the unsolicited Key-based Pairing write when not in pairing mode, whereas a vulnerable Provider processes it and pairs with the attacker.

obtain the public Anti-Spoofing key using the Model ID and derive a fresh AES key from an ephemeral secp256r1 key pair. The pairing request m is then built using the Provider's current BLE address a_1 and nonce s.

After connecting to P, enabling notifications, and sending a write request, the Provider replies with its BR/EDR address. Instead of proceeding, we resend m. If the Provider again responds, it indicates that reused nonces are not detected within the same connection. This behaviour is not strictly non-compliant, as the specification allows vendors to rely either on nonce tracking or on address rotation.

We then disconnect to let the Provider rotate its random BLE address from a_1 to a_2. Since m embeds a_1, it should be invalid in a new connection with a_2. If the address is not rotated, m can be replayed after reconnection. Even when rotation occurs, replaying messages over a new unauthenticated BLE link remains possible, as Fast Pair provides no link-layer integrity.

4.5. Invalid Curve Check

Fast Pair uses ECDH on secp256r1 to establish a session key. In contrast to TLS, which includes a negotiation phase for selecting the curve [17], Fast Pair fixes the curve and requires the Provider to compute ECDH using the Seeker's public key. If the Provider does not verify that this public key lies on secp256r1, it becomes susceptible to the invalid curve attack [18]. Such attacks have been demonstrated on TLS-ECDH [18], Bluetooth [19], and Wi-Fi [20].

To test whether a Provider performs this validation, we follow the method of [18]. We select a public key on a different curve such that any ECDH computation results in one of three possible session keys. Using each candidate key, we craft a pairing request with an encrypted payload derived from that key and send it as a write request. If the Provider replies to any of these messages, it indicates that it accepts public keys not on secp256r1, and therefore does not validate the curve membership of the Seeker's key.

5. *WhisperPair*: A Family of Practical Attacks

This section uses the failures exposed by the conformance tests in Sect. 4 to develop *WhisperPair*. We demonstrate two practical attacks that follow from non-enforcement of the pairing state predicate in Google Fast Pair [1]. Furthermore, we examine a set of secondary abuse patterns that emerge from the same root cause or from related robustness issues. We show what happens when the security compliance tests discussed before fail, and what the security and privacy implications are for hundreds of millions of users. Our aim is to show how a single violated state predicate results in serious privacy and security risks. Throughout, we assume the threat model presented in Sect. 4.1. To illustrate WhisperPair in practice, Fig. 3 shows Eve (the attacker) injecting an unsolicited pairing request to Bob (the earbuds) while they are still connected to Alice (the user); the earbuds accept the request, establish a new Bluetooth session with the attacker, and can then be operated without the owner's knowledge. Later, in Sect. 6, we explain how we demonstrated and evaluated our attack in detail.

5.1. Forced Audio Takeover Outside Pairing Mode

The primary consequence of *WhisperPair* is that a nearby unauthorised host can compel a Provider to attach while the Provider is in steady state. The specification requires Providers to reject Key-based Pairing writes unless they are in pairing mode [1], yet most certified devices still process such writes when advertising account-data beacons rather than pairing beacons. Once the Provider accepts the write, it completes ECDH with the attacker's public key, derives the session key, and transitions to BR/EDR attach-

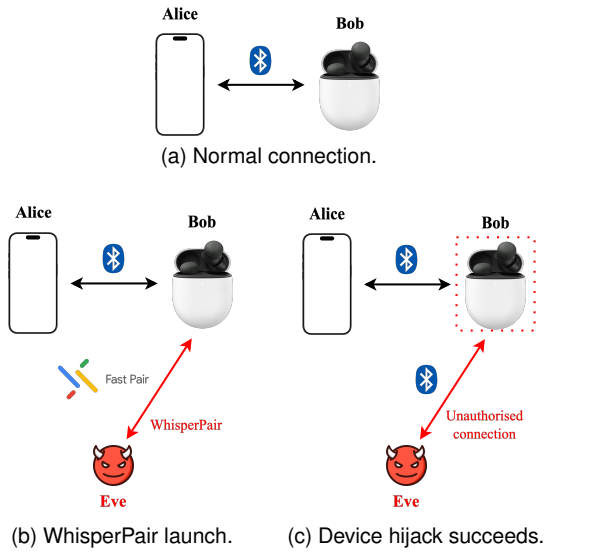


Figure 3. The WhisperPair hijack. (a) A benign Bluetooth session: Alice's phone streams audio to her earbuds. (b) Eve exploits Fast Pair, launching WhisperPair to initiate an unsolicited pairing with the same earbuds. (c) The attack succeeds: Eve completes an unauthorised session and takes control of the earbuds.

ment. The attacker then completes the Bluetooth numeric comparison stage and the accessory bonds to the attacker's Seeker.

The immediate effect is an abrupt hijack of the audio device. The legitimate host loses the stream instantly or whenever the attacker requests a switch (*AudioSwitch*), often with only a brief audible cue. Control returns only after a delay or manual intervention. The adversary can start playback immediately and, for many models, gains microphone access without any prompt on the victim's handset. The entire attack completes within seconds, as measured in Sect. 6, and requires only commodity equipment, making it practical in environments such as trains, classrooms and open offices.

The harm is not limited to availability. Playback control enables the adversary to inject audio at sensitive moments, suppress alarms or spoken instructions, or trigger actions in applications that rely on auditory cues. When microphones become available immediately after bonding, the attacker can escalate from nuisance to privacy harm by recording ambient speech near the wearer without any interaction with the victim's smartphone. The takeover also undermines core user expectations of pairing. *Users assume that a stable accessory will not attach to a new host unless they explicitly enter pairing mode.* The observed behaviour violates this assumption in a way that is difficult for users to detect.

5.2. Covert Account Binding & Location Tracking

The second *WhisperPair* attack appears when an accessory implements the Android device-finding extension for GFPS [1]. This extension allows accessories to be associated with a Google account for crowdsourced location tracking via the Find Hub network. The association is represented by an account key. In principle, this supports owners who wish to locate misplaced devices. In practice, non-enforcement of the pairing state predicate allows a nearby adversary to act as the first writer of that account key, to be recognised as the device owner, for accessories whose owners are not Android users or who have never enrolled the device, such as iOS users. Once the accessory accepts the attacker's account key, it becomes logically attached to the attacker's account.

This leads to a qualitatively different harm from audio takeover. Covert binding turns a brief proximity attack into persistent location tracking and stalking that continues until the accessory is factory-reset. The owner receives little or no notification because the operation targets the accessory rather than the phone. In our experiments, tracking appeared soon after binding, survived normal case open and close cycles, and persisted across daily use. Even when a delayed cross-platform anti-stalking alert eventually appears, it identifies the victim's *own* accessory, which many users may dismiss as a benign glitch.[††] Across multiple models, our results show how a convenience feature can be repurposed against users who never chose to participate in the ecosystem.

[††]In our experiment, an attacker tracked a victim for 48 hours before the anti-stalking alert was triggered.

5.3. WhisperPair Attack Implications

The weaknesses exposed by *WhisperPair* directly affect hundreds of millions of users who rely on Fast Pair compatible accessories across Android, iOS, and other platforms. In this subsection, we outline the resulting harms, showing how a seemingly small design oversight in the pairing layer scales into widespread security and privacy risks.

Short-window ambient audio capture. A practical adversary does not require continuous control to cause privacy loss. We can trigger a recording intent, and capture ambient speech from the environment around the wearer, connect, disconnect and again reconnect in a short window. This attack is subtle because the accessory often auto-reconnects to the legitimate host extremely quickly, which reduces suspicion, yet the adversary accumulates intelligible fragments. The result is a privacy harm in shared environments, including meeting rooms and public transport, and does not depend on decrypting any handset traffic.

Social engineering through voice injection. A takeover during an active call or meeting lets an adversary inject brief prompts that influence the victim's next action, such as rejoining a meeting, approving a multifactor request, or calling back a contact. Because the audio comes through the victim's trusted accessory, it appears legitimate. The attacker learns no session keys, but the integrity of the audio channel is compromised, creating a genuine security risk in settings where users rely on spoken cues to make decisions.

Crowd-level denial of audio. With only commodity hardware and minimal timing, a single adversary can disrupt many users in quick succession. Walking through a lecture hall or train carriage, the attacker can trigger repeated short takeovers, creating a low-cost denial of service that affects large groups with little risk of detection and does not rely on laboratory conditions.

Audio switching and multipoint manipulation. Modern accessories support multipoint attachment and audio switching across several hosts. When the pairing-state predicate is not enforced, an adversary can join as an additional host and preempt the owner by requesting switches at chosen times. In Sect. 6 we examine whether switching can be forced from idle, how often an attacker can pre-empt playback, and whether multipoint support prolongs the hijack state. These results show how convenience features amplify the underlying weakness.

Stalking and covert tracking at scale. The most severe implication is cross-ecosystem surveillance. By binding a victim's accessory to an attacker account, *WhisperPair* turns the victim's own hardware into a tracker within Find Hub. This affects users outside the Android ecosystem, including iOS users who never opted into Google's device-finding features. The tracking is silent, persists across normal use, and stops only after a factory reset. The attack takes only a few seconds without any prior assumptions or interactions. An attacker can sit in a crowded place (such as airports or train stations or inside public transport) and launch WhisperPair to immediately track numerous victims in a given Bluetooth range, without them knowing.

Hearing safety and situational awareness. Finally, we consider harms that touch on safety. A forced takeover permits the adversary to inject high-volume audio or to toggle noise control modes such as active noise cancellation or transparency, with obvious safety consequences in traffic and other settings.

From single-device hijack to ecosystem risk. Taken together, these attacks show that one violated predicate at the core of GFPS leads to harms spanning nuisance, denial of service, privacy loss, and safety risk. They succeed without breaking link encryption or privileged OS components, and they appear across vendors and chipset families. Certification is therefore central: if validation does not exercise the required state predicate, vendors will ship devices that pass functional testing yet fail to meet essential security expectations. In Sect. 7.1 we show how current validation misses these behaviours and how modest changes to firmware checks and test procedures could enforce the intended boundary.

6. Methodology and Evaluation

Fast Pair devices must pass certification before reaching consumers. Because small implementation errors can have substantial impact, we conducted a large-scale study of certified accessories using the tests introduced in Sect. 4. This section describes our experimental setup, the procedure for executing each test, and the results obtained from our evaluation.

6.1. Experimental Setup

We implemented *WhisperPair* on a Raspberry Pi 4, using the BlueZ Linux API to read and write GATT characteristics. Alongside the compliance tests, we implemented practical attacks to assess whether a device can be hijacked or tracked by an adversary.

6.1.1. Collecting Device Information. Fast Pair accessories advertise their Model ID, but pairing requires the corresponding Anti-Spoofing public key. By intercepting Android network traffic, we identified the API used to retrieve Fast Pair metadata, including public keys and internal product names. Although the API applies IP-based rate limiting, we bypassed it by issuing requests from multiple IP addresses, enabling us to compile a complete list of certified models. While the API only returned data for models marked "published", we observed one device that had not yet been publicly announced.

6.1.2. Unauthorised Pairing. To test whether a device can be hijacked, we implemented the pairing state predicate check from Sect. 4.3. Before each attempt, the accessory was connected to a normal Android device and not in pairing mode. We scanned for nearby BLE devices advertising Fast Pair data, selected a target, opened a connection, and read the Model ID characteristic. As many devices did not expose this characteristic correctly, our implementation also supports manual entry. Because most accessories advertise their model name, an adversary can alternatively infer the Model ID using the complete list of certified Fast Pair devices.

Once the Model ID was known, we followed the procedure in Sect. 4.3 to initiate unauthorised pairing. We sent a standard pairing request containing an ephemeral public key. A vulnerable Provider replies with its BR/EDR address, enabling us to start pairing. We completed the procedure by writing the numerical passkey to the appropriate characteristic. After BR/EDR pairing succeeded, we wrote an account key to the Provider, allowing us to generate the authentication codes needed for particular Fast Pair extensions.

6.1.3. Audio Switching. Unauthorised pairing does not always disconnect the victim. Providers that support multipoint can maintain simultaneous connections to several hosts. Behaviour varies by model: some switch to the attacker immediately, while others change hosts only after the victim pauses audio. Even without multipoint, a Provider may keep the victim's connection active. The *Audio Switch* extension further allows supported Providers to switch between Seekers automatically based on inferred user actions, which an attacker can exploit once paired.

This extension enables supported providers to automatically switch between seekers based on user actions. After pairing with a Provider, we tested whether the Audio Switch extension can be abused. Each Audio Switch message must include a Message Authentication Code (MAC) derived from a session nonce and an account key. Using the account key written earlier, we generated valid MACs and were able to force the Provider to disconnect from the victim. An attacker can also trigger a reconnection to the victim. These transitions are indistinguishable from ordinary Bluetooth disconnects because the interruption lasts only a few seconds.

6.1.4. Find Hub Provisioning. To test whether a Provider can be covertly tracked, we attempted to enrol vulnerable devices into the Find Hub network. Enrolment is only possible if the attacker's account key is marked as the Owner's Account Key (OAK), which corresponds to the first account key written to the Provider and can only be cleared by a factory reset. Android devices automatically write an account key after pairing, preventing later keys from being marked as the OAK. However, users who never pair their accessory with an Android device leave the first-writer position available to an attacker.

For each device supporting Find Hub, we factory-reset the Provider and initially paired it to a non-Android device. We then carried out the attack from Sect. 6.1.2, but completed the BR/EDR pairing using an Android device. Because the Provider is not BR/EDR discoverable in this workflow, we used an Android API to pair using the Provider's BR/EDR address. After pairing, the Android device automatically prompted us to add the accessory to Find Hub.

TABLE 1. EVALUATED DEVICES WITH THEIR BLUETOOTH CHIPSETS AND THE OUTCOME OF THE SECURITY EVALUATION. SYMBOL LEGEND: ○ NOT VULNERABLE, ● VULNERABLE, × FEATURE NOT SUPPORTED / NOT APPLICABLE. FOR SELECTED COLUMNS, THE SYMBOLS CONVEY EXTRA DETAIL. *Hijack*: ○ BLOCKS UNSOLICITED PAIRING, ◐ PAIRS BUT AUDIO STAYS WITH THE VICTIM, ● FULL TAKEOVER (FORCE DISCONNECTS THE VICTIM); *Nonce Reuse*: ○ VERIFIES NONCES, ◐ ACCEPTS A REPEATED NONCE ONLY WITHIN THE CURRENT SESSION, ● ACCEPTS ANY REPEAT. *Time*: DURATION OF THE HIJACKING PROCEDURE IN SECONDS.

Device Information			**Chip Information**		**WhisperPair Attack**			**Post-Hijack**		**Replay**
Manufacturer	Model	Type	Manufacturer	Model	Hijack	Find Hub	Time [s]	Mic	Switch	Nonce Reuse
Apple	Beats Solo Buds	Earbuds	MediaTek	MT2827SA	○	×	×	○	×	●
Google	Pixel Buds Pro 2	Earbuds	Google	Tensor A1	●	●	6.89	●	●	○
Jabra	Elite 8 Active	Earbuds	Airoha	AB1585	●	×	32.01	●	×	○
JBL	Tune Beam	Earbuds	Bestechnic	BES2600Z	●	×	6.91	●	×	◐
Marshall	MOTIF II A.N.C.	Earbuds	Airoha	AB1588Q	●	×	9.49	●	●	●
Nothing	Ear (a)	Earbuds	Bestechnic	BES2600Z	●	×	38.80	●	●	◐
OnePlus	Nord Buds Pro 3	Earbuds	Bestechnic	BES2700ZP	◐	×	10.19	●	×	◐
HP	Poly VFree 60	Earbuds	Qualcomm	QCC5171	○	×	×	○	×	○
Redmi	Buds 5 Pro	Earbuds	Airoha	AB1577SA	●	×	8.32	●	×	◐
Soundcore	Liberty 4 NC	Earbuds	Realtek	RTL8976	●	×	15.27	●	×	●
Sony	WF-1000XM5	Earbuds	MediaTek	MT2833	◐	●	9.43	●	●	●
Audio-Technica	ATH-M20xBT	Headphones	Qualcomm	QCC3056	○	×	×	○	×	◐
Bose	QuietComfort Ultra	Headphones	Qualcomm	QCC5181	○	×	×	○	×	○
JBL	Live 775 NC	Headphones	Qualcomm	QCC5171	◐	×	7.62	●	×	◐
Marshall	Major V	Headphones	Airoha	AB1588	●	×	11.70	●	×	◐
Sonos	Ace	Headphones	Qualcomm	QCC5181	○	×	×	○	×	○
Sony	WH-1000XM4	Headphones	MediaTek	MT2811	●	×	9.69	●	×	●
Sony	WH-1000XM5	Headphones	MediaTek	MT2822	●	●	12.38	●	●	●
Sony	WH-1000XM6	Headphones	MediaTek	MT2833	◐	●	12.94	●	●	◐
Sony	WH-CH720N	Headphones	MediaTek	MT2822	●	×	7.46	●	×	●
Bang & Olufsen	Beosound A1	Speaker	Qualcomm	QCC5127	○	×	×	○	×	○
Jabra	Speak2 55 UC	Speaker	Qualcomm	QCC3056	○	×	×	○	×	○
JBL	Clip 5	Speaker	Actions	ATS2835	●	×	36.19	×	×	●
JBL	Flip 6	Speaker	Actions	ATS2835	○	×	×	×	×	○
Logitech	Wonderboom 4	Speaker	Qualcomm	QCC3040	●	×	11.96	×	×	◐

Once enrolled, the Provider starts advertising ephemeral identifiers that nearby Android devices scan and report to Google. As the Provider has been added to the attacker's Google account, these location reports are forwarded to the attacker, enabling covert tracking of the victim.

6.2. Results

To cover a broad range of brands and device categories, we tested earbuds, headphones and speakers from multiple vendors. Table 1 lists the 25 accessories evaluated, representing 16 manufacturers, which, to the best of our knowledge, encompass all major audio brands that currently offer Fast Pair support. Because GFPS functionality also resides in the Bluetooth system-on-chip, we identified the underlying 17 distinct radio hardware chipsets from seven different vendors.

With respect to the attacks, we evaluated several dimensions and grouped them by the capabilities they expose.

WhisperPair attack:

- *Hijack:* can the attacker take control of the accessory?
- *Find Hub:* on accessories compatible with Google's Find Hub network, can the attacker bind the accessory to their own Google account for covert location tracking?
- *Time:* how long does the hijacking procedure take in practice?

Post-hijack capabilities:

- *Mic:* after a hijack, can the attacker access the microphone?
- *Switch:* on accessories that support audio switching, can the attacker move playback between the legitimate host and the attacker's device?

Independent protocol checks:

- *Nonce Reuse:* does the accessory correctly enforce nonce freshness, i.e. does it reject repeated nonces as discussed in Sect. 4.4?
- *Invalid Curve:* does the accessory verify that the peer's public key lies on the correct elliptic curve, as outlined in Sect. 4.5?

The results appear in Table 1, except for the Invalid Curve check, which is omitted because every device passed it.

Of the 25 accessories tested, 17 (68%) fail the pairing-state predicate, as shown by the half- or full-circle symbols in the Hijack column. Although multipoint behaviour varied, all non-multipoint devices disconnected from the victim after our attack. On every hijacked device we could access the microphone, yielding a 100% success rate. Google's Find Hub feature remains new and is not yet widespread [21]: only four evaluated models support it, and all four could be silently bound to the attacker's account for covert tracking. Audio switching is available on six devices and was exploitable on all six. For each vulnerable model, we ran the hijack five times and averaged the takeover time; these values appear in the Time column. All measurements were taken at a separation of 14 m, matching our test hall. Most devices completed the attack in under 15 s at this distance, confirming practical feasibility. Repeating the experiments at shorter distances produced identical timings and success

rates, so Table 1 reports only the 14 m results. Turning to the protocol checks, 17 devices (68%) mishandle nonce validation: six accept a reused nonce within a single connection (half-circle), and another six accept any repeated nonce (full-circle). All devices passed the invalid curve test.

The results show clear patterns. Whenever a device is vulnerable to hijack, every post-hijack capability it exposes (microphone access, audio switching, and Find-Hub binding) is also exploitable, demonstrating full attacker control. The most robust devices use Qualcomm chipsets, suggesting a more faithful implementation of the GFPS specification. Price, however, does not correlate with security: Sony's WH-1000XM6 flagship headphones were vulnerable to all attacks, whereas HP's Poly VFree 60, at less than half the cost, resisted them entirely. Overall, the results reveal substantial variation across brands, models and cost.

7. Discussion

This section integrates our empirical results with the research questions in Sect. 2.2, to show the root cause of the problems. We show that the observed failures are not isolated bugs but symptoms of systemic weaknesses that affect hundreds of millions of devices, outline their practical and societal consequences, and identify engineering and policy levers that can restore security.

7.1. Compliance Chain Failures in Fast Pair

Achieving Fast Pair compliance requires vendors to follow a multi step adoption process defined by Google [1]. Our evaluation shows that critical weaknesses arise at several points in this workflow, culminating in severe security and privacy failures, including forced device takeover and covert location tracking (Sect. 5). In this subsection, we pinpoint where the compliance chain breaks. We explain why certified accessories still accept unauthorised Key-based Pairing outside pairing mode, linking this to upstream design and integration issues (**RQ1**), and we identify the observable behaviours that form a practical conformance oracle while showing why current validator artefacts do not detect these deviations (**RQ4**).

Implementation stage. The first failure point is the implementation phase. We confirmed through our technical meetings with Google that conformance breaks at two layers: (i) accessory vendors, including major brands such as Sony, Google, and JBL, frequently omit the explicit pairing mode check, and (ii) Bluetooth chipset manufacturers ship firmware that diverges from the specification. Not every deviation yields an immediate vulnerability, but the breadth of errors is concerning. Since such mistakes are common, the key question is whether the specification can enforce correct behaviour in practice. These findings identify upstream causes of non-enforcement, addressing **RQ1**.

Validation stage. A further weakness lies in Google's Validator App, an Android tool on the Play Store that vendors must run before certification [22]. Marketed as an app that "*validates that Fast Pair has been properly implemented on a Bluetooth device*", it produces pass reports that vendors submit as evidence of compliance. In practice, the Validator allowed the mis-implemented accessories we examined to pass, permitting security and privacy flaws, including the covert tracking attack. Accepting a Key-based Pairing write while the Provider advertises account data in steady state is an observable oracle that public validation artefacts failed to test, addressing **RQ4**. We could not run the Validator ourselves because access is restricted to manufacturers. Google informed us that it has identified these shortcomings and has patched the Validator to reject non-compliant devices.

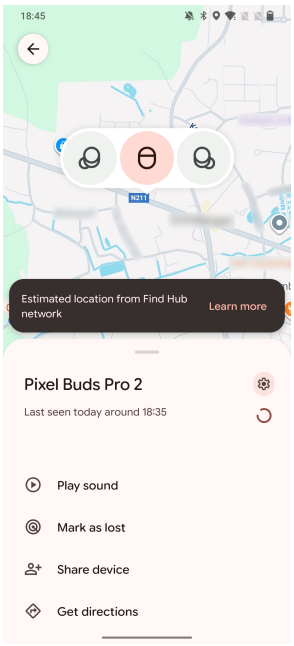
Certification stage. The final and arguably most serious breakdown occurs during Google's official certification process [1]. Vendors must submit pass reports and physical device samples to a Google-approved laboratory for conformance testing before GFPS can be used in production. Google states that its team will issue a defect report if certification fails. However, several mis-implemented accessories still passed this stage. In our discussions, Google noted that certification tests and procedures have since been updated to address this problem. The absence of this negative test explains its prevalence across vendors and chipsets, and identifies a clear testability gap, addressing **RQ4**.

7.2. Security and Privacy Impact on Users and Ecosystem

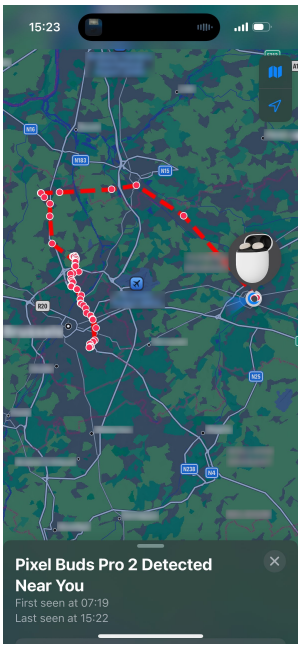
Not a one-off defect. The predicate failure is not confined to a single vendor or chipset. We observed identical behaviour across multiple devices and chipset families, indicating a shared upstream origin in reference stacks or integration guidance. These devices passed vendor quality assurance and certification, which means the failure survived multiple layers of scrutiny. This convergence and survivability demonstrate a systemic process failure rather than individual developer error, addressing **RQ1**.

User-centred harm and cross-ecosystem impact. When the boundary collapses, the consequences are immediate and personal. A nearby adversary using commodity hardware can attach in seconds, seize audio output, and activate microphones without consent. As measured in Sect. 6.2, time-to-hijack across five trials per device ranged from 6 to 35 seconds, with a median of 10 seconds, and succeeded at 14 m in our testbed, well within ordinary Bluetooth range.

The covert account binding attack is more severe. A victim's accessory can be silently enrolled under an attacker's Google account and begin reporting its location through the crowdsourced *Find Hub* network. The victim's own hardware becomes a tracker that sends continuous location reports to an unauthorised account, without any planted device or visible cue. Anti-stalking safeguards exist, yet are too slow and ambiguous in this setting. The joint Apple–Google framework [23], [24] defines a set of guidelines and protocols enabling mobile platforms to detect unwanted trackers. When such a warning is triggered, it names the victim's own accessory as the tracker, which many users may dismiss as a benign glitch. We reproduced this case with an iPhone user wearing Pixel Buds Pro 2.



(a) Attacker's Google Find My dashboard with hijacked buds' live location.



(b) Victim's iPhone unknown tracker alert showing the buds' past movements.‡‡

Figure 4. Attacker and victim perspectives after the attacker enrols the victim's Pixel Buds in the attacker's Google account, covertly turning the buds into a tracker within the Find Hub network.

After hijacking the buds, the attacker linked them to their Google account. Figure 4 shows both views: the attacker's dashboard displayed the live location immediately (Fig. 4a), whereas the victim's iPhone produced an unknown tracker alert only after roughly 48 hours (Fig. 4b). During this window, the buds logged movements across several cities, illustrating how much tracking can occur before any warning appears. The outcome is persistent, stealthy tracking and stalking without any added hardware, and it affects users outside Google's platform. iOS and other non-Google users are exposed through the same backend. These findings address **RQ2** and **RQ3**: a convenience-focused design has created a cross-ecosystem surveillance vector that ordinary users cannot detect, or revoke in time.

Ecosystem implications. Security fails on both sides of the compliance workflow. Vendors introduce implementation flaws, and the platform's validation and certification stages have, until recently, not exercised the adversarial sequence that reveals them. This two-sided lapse leaves the entire Fast Pair ecosystem vulnerable, and it shows that strong guarantees require rigorous vendor engineering *and* reliable, defence-in-depth testing by the platform owner.

‡‡The alert is not simultaneous. It appeared after 48 hours in our experiments.

7.3. Mitigation and Deployability

Remediation must work for both new and legacy devices and must be deployable at scale. Immediate steps include enforcing the pairing state predicate directly in firmware and expanding validator coverage to explicitly test this invariant. Although the specification already defines the predicate as a mandatory conformance test, its current phrasing does not guarantee that vendors implement or exercise it in practice. Strengthening the conformance language would reduce ambiguity and close the gap between specification and deployment. Platform-level defences can further assist by alerting users when an already-bonded accessory attaches to a new host or when reconfiguration occurs unexpectedly. Where firmware updates are infeasible, minimal on-device gestures before accepting a new host would restore explicit consent without significant usability loss.

To eliminate the flaw by design, we propose *IntentPair*, a lightweight cryptographic modification that binds pairing intent directly into the key schedule. This ensures that pairing succeeds only when the device is in pairing mode or when the requester proves prior ownership. IntentPair preserves wire compatibility, adds negligible computational cost, and supports asymmetric deployment across existing ecosystems. The full design and evaluation are presented in Sect. 8. These measures jointly address **RQ5**: they provide a practical path to enforce pairing intent, strengthen certification coverage, and restore the consent boundary without diminishing the usability benefits that motivated Fast Pair.

7.4. Generalisability and broader lessons

Fast Pair's weaknesses exemplify a recurring pattern in consumer IoT design: usability features that bypass explicit consent eventually erode the guarantees inherited from lower layers. Any ecosystem that introduces account-level functionality, automatic switching, or cross-transport transitions must define and validate strict state predicates for reconfiguration. Absent this, devices converge on "happy path" behaviours that delight users during demonstrations but expose them in daily life. The same principle applies to Matter, Apple's ecosystem, and future convenience-first standards, addressing **RQ6**.

8. *IntentPair*: Binding Pairing Intent Cryptographically

We formalise our mitigation and propose *IntentPair*, a lightweight cryptographic hardening that binds user pairing intent into key derivation so that any handshake lacking consent fails by design. Rather than introducing a new pairing protocol, IntentPair strengthens the existing Fast Pair workflow by enforcing fail-closed authorisation at the key derivation stage.

8.1. Threat Recap and Notation

Fast Pair derives a session key from an ECDH exchange even when the device is not in pairing mode. Let $\mathbb{G}$

be a prime-order group with scalar–point multiplication $\cdot$. The Provider holds $(\mathsf{sk}_P, \mathsf{pk}_P) \in \mathbb{Z}_q \times \mathbb{G}$ and the Seeker $(\mathsf{sk}_S, \mathsf{pk}_S)$. Both compute

$$z = \mathsf{ECDH}(\mathsf{sk}_S, \mathsf{pk}_P) = \mathsf{ECDH}(\mathsf{sk}_P, \mathsf{pk}_S), \tag{1}$$

encoded as a bit string of length κ. Let T denote the transcript of capability bits and pre-pairing messages, and derive

$$K = \mathsf{KDF}(z \parallel \mathsf{T}). \tag{2}$$

Because the predicate "in pairing mode" is absent, pairing may succeed without user intent.

Design Goal: Pairing should succeed only when intent or ownership is cryptographically verifiable. Hence, the predicate check is moved inside the key schedule: without pairing mode or ownership proof, the handshake fails automatically.

8.2. Construction: Intent Nonce Bound to the KDF

On entering pairing mode, the Provider samples a nonce

$$I \xleftarrow{\$} \{0,1\}^{\lambda}, \tag{3}$$

advertised only while pairing is active and discarded after use. Using HKDF with 32-byte output, both sides derive

$$\mathsf{PRK} := \mathrm{Extract}(\mathsf{salt} = I, \mathsf{IKM} = z), \tag{4}$$

$$K := \mathrm{Expand}(\mathsf{PRK}; \mathsf{info} = \mathrm{GFPS\text{–}KbP} \parallel \mathsf{T}; \mathsf{L} = 32). \tag{5}$$

Both obtain the same K only if the pairing mode was entered; otherwise, I is absent and the derivation fails. *IntentPair* modifies the original Fast Pair key derivation by adding context binding through T, as Fast Pair lacks session binding.

Correctness. In pairing mode both sides know (z, I) and derive identical K; otherwise the Provider lacks I.

Security intuition. I binds entropy to the pairing window. Security rests on I existing only during legitimate, user-initiated pairing.

Implementation note. Vendors may instead expose $S = H(I)$ and use $\mathsf{salt} = S$, which is equivalent if S is created only when pairing begins.

8.3. Authenticated Re-pair Path

For bonded devices sharing an account key ak, re-pairing proceeds only after ownership proof. The Provider issues $c \xleftarrow{\$} \{0,1\}^{\lambda}$; the Seeker replies

$$\tau = \mathsf{MAC}_{\mathsf{ak}}(c \parallel \mathsf{T}). \tag{6}$$

If τ verifies, the Provider derives K with $\mathsf{salt} = H(\mathsf{ak} \parallel c)$, binding the session to authenticated ownership and blocking unsolicited takeover.

8.4. Fail-Closed Attachment Check

The Provider initiates BR/EDR attachment only if AEAD verification under K succeeds:

$$\mathsf{AEAD\_Verify}(K, \mathsf{AD}, \mathsf{ct}, \mathsf{tag}) = 1. \tag{7}$$

Otherwise, it returns an error and remains in the BLE state, enforcing the predicate cryptographically.

8.5. Relation to Established Practice

The construction aligns with patterns in other security protocols. In TLS 1.3 [25], exporter keys depend on both the Diffie–Hellman secret and a hash of the transcript, causing any omission or reordering to abort the handshake. In FIDO2 [26] and WebAuthn [27], authenticators sign a challenge together with the relying-party identifier, binding user intent to the origin. Bluetooth LE Secure Connections [10] similarly includes pairing nonces in link-key derivation. Our approach extends this principle to Fast Pair: we embed the predicate directly into the key derivation, resolving a design flaw at its source. We follow the principle that *if a problem can be solved at the top, it should be solved at the top*, preventing logic-level omissions from propagating downward into vendor firmware or certification.

8.6. Security Properties

Under standard ECDH and HKDF assumptions in the random-oracle model:

- *Intent binding.* For any probabilistic polynomial-time adversary interacting with a Provider not in pairing mode and without a valid ownership key, the probability of both parties deriving the same K is negligible in λ.
- *KDF robustness.* The derived key K remains pseudorandom even if I (or S) is observable, as I only influences the HKDF salt input.
- *Replay resistance.* Because I is short-lived and c is freshly sampled on re-pair, replayed transcripts fail owing to mismatched salts and challenges.

8.7. Security Proofs

Formal game-based proofs of *IntentPair* under standard cryptographic assumptions appear in Appx. A.

9. Related Work

9.1. Bluetooth security and pairing weaknesses

A long line of results shows that seemingly small design choices in Bluetooth pairing can undermine core guarantees. KNOB demonstrated that legacy BR/EDR key negotiation could be forced to one byte of entropy, enabling practical brute force of link keys [2]. BIAS showed that classic authentication can be bypassed to impersonate devices without prior pairing [3]. BLURtooth revealed that cross-transport

key derivation between BLE and BR/EDR can overwrite stronger keys with weaker ones, enabling unauthorised access [4]. BLUFFS further showed that flaws in session key derivation break forward and future secrecy, allowing message recovery and impersonation across sessions [5]. Traceability attacks further show that persistent identifiers and address management in BLE enable both active and passive device tracking despite intended privacy protections [28]. Various other insecurities consolidate these lessons and emphasise how compatibility features and layered complexity repeatedly erode security boundaries [11].

Our work differs in scope and locus. Prior attacks primarily target the Bluetooth core specification or controller behaviour. We instead study *Google Fast Pair* as an application-layer extension on top of Bluetooth, focusing on both protocol-level enforcement failures and their ecosystem-level consequences. We show that when pairing relies on fallible state checks instead of cryptographic proof of user intent, flaws persist across vendors and chipsets despite correct Bluetooth implementations.

9.2. Usability, security, and commercial ecosystems

The tension between usability and security is well-documented in authentication and pairing. Bonneau *et al.* present a comparative framework for evaluating authentication schemes, underscoring the limits of human-centred channels and the risks of convenience-driven design [29]. In home IoT, vendor diversity, opaque firmware, and weak validation expose devices despite sound specifications [30]. Commercial ecosystems show the same trend: Zigbee has demonstrated how usability, legacy allowances, and certification gaps render devices exploitable [31], while Matter repeats this lesson at scale: strong cryptography but exploitable behaviour due to mis-implementation and ambiguous guidance [32].

Fast Pair fits this pattern. It is a usability layer promising one-tap onboarding, account synchronisation, and frictionless pairing. Our findings show that risk stems not only from cryptographic design but from weak policy enforcement. In Fast Pair, the predicate "device is in pairing mode" remains application logic rather than a cryptographically bound condition, propagating failures through implementation, validation, and certification to create systemic exposure.

9.3. Fast pairing and accessory ecosystems

Apple's MagicPairing shows how vendors conceal protocol complexity behind account-based services, and research reveals that even mature ecosystems can have exploitable flaws [6]. Two strands of prior work on Google's proximity services are directly relevant. Analyses of Nearby Connections show that closed-source proximity stacks expose attack surfaces through application-level shortcuts and ambiguous state handling [7]. Studies of Google's crowdsourced tracking ecosystem demonstrate how account binding and device enrolment can be abused for large-scale tracking when an attacker can register or influence device state [8]. These works target adjacent layers: Nearby focuses on connectivity primitives, while Find Hub and related studies examine how synchronised devices enable location inference.

Our work unifies these perspectives. While prior attacks [8] required planting an external tracker, our attack allows a victim's accessory to be silently enrolled and converted into a covert tracker within Google's Find Hub network. This attack is stealthier and more persistent as it leverages legitimate account synchronisation rather than extra hardware or user actions. Method-confusion and capability-coercion attacks likewise exploit assumptions about device state and user prompts [33], and our results show that such abuse persists at the Fast Pair layer unless pairing intent is cryptographically enforced.

To the best of our knowledge, there has been no in-depth analysis of Fast Pair itself. Prior Bluetooth studies have focused on lower-layer protocols (LMP, LL, SMP) or transport-agnostic threats such as link key reuse and session key derivation flaws [2], [3], [5]. Vendor-specific pairing extensions, including Google Fast Pair, have been treated largely as usability abstractions rather than as security-critical protocols, a gap that this work directly addresses.

10. Conclusion

Pairing is a security boundary. Users reasonably expect that their accessories cannot be commandeered unless they intentionally enter pairing mode and confirm a prompt. This work revisits the security foundations of Google Fast Pair and exposes a systemic weakness at the intersection of specification guidance, vendor implementation, and certification oversight. Through extensive empirical testing across multiple commercial devices, we demonstrated that the Key-based Pairing mechanism can be silently abused to seize control of accessories, access microphones, and register devices to the attacker's Find Hub account, thereby silently stalking and tracking the users using their own devices. Our analysis revealed that these failures are not isolated implementation oversights, but result from a deeper structural fragility in the ecosystem's compliance chain. The interplay between permissive specification clauses, inconsistent enforcement across vendors, and incomplete validation checks collectively undermines the intended pairing predicate and allows unauthorised re-pairing to persist at scale. To address these flaws, we proposed *IntentPair*, a minimal yet effective modification of the Fast Pair protocol that cryptographically binds the user's pairing intent to the derived session key. By conditioning key derivation on an authenticated proof of ownership, IntentPair prevents re-pairing attacks without imposing new hardware dependencies or altering usability. Beyond its immediate applicability to Fast Pair, this study highlights a broader lesson for the design of convenience-first pairing ecosystems. Security-by-certification is only as strong as the verifiable enforcement of its invariants. Ecosystem-level resilience requires not only cryptographically sound protocols, but also testable oracles that expose non-compliance early in the validation pipeline.

Acknowledgements

This work was supported by the Flemish Government through the Cybersecurity Research Program, grant number VOEWICS02. The authors thank all individuals who kindly lent us their devices and agreed to let them be used for testing. We are also grateful to Google for the constructive collaboration during the disclosure process and for the productive discussions and meetings that helped address the reported issues. Seppe Wyns carried out a substantial part of his contribution to this work during his summer internship at COSIC, KU Leuven, in 2025.

Ethics Considerations

We conducted this study in accordance with established ethical practices for security research and coordinated disclosure. We reported the vulnerabilities to Google on 14 August 2025 and, by mutual agreement, adopted a 150-day disclosure timeline to allow sufficient time for internal review, partner outreach, and remediation. Google acknowledged the issues, assigned CVE-2025-36911, and assessed the findings as *critical* with *high quality* impact under their internal severity guidance [9]. Because Fast Pair is a Google-defined protocol, Google operates the certification and validation process, and one attack path involves Google's Find Hub network, disclosure through the protocol owner was necessary for centralised mitigation across vendors. Google informed us that they had issued an urgent security notice to affected vendors and had begun patch coordination. We also discussed our proposed mitigation, *IntentPair*, with Google as part of mitigation planning.

We deliberately constrained the scope of our experimental evaluation. We tested 25 Fast Pair-certified commercial products spanning multiple vendors, chipsets, and reference firmware stacks, and we selected devices to maximise chipset and vendor diversity rather than total count. Our impact discussion is therefore grounded in direct testing on these devices together with the observation that Fast Pair logic is often reused across shared SDKs, chipset integrations, and certification workflows. Although our tooling could in principle support broader enumeration of nearby Fast Pair advertisements, we did not conduct opportunistic or large-scale public scanning. All testing was limited to devices that we ethically owned, purchased specifically for the study, or borrowed with explicit permission.

All experiments were conducted with informed consent and full authorisation. The tested devices were either bought new for this study or voluntarily loaned by colleagues, friends, and family members, all of whom were informed of the research goals and methodology. Before each experiment, participants explicitly consented to the temporary pairing, account-key insertion, and other steps required to demonstrate the hijacking and tracking effects. After each test, devices were factory-reset in the owner's presence to verify removal of residual associations, including account-binding state where applicable. We did not target unauthorised third-party devices or users, and we did not collect data beyond what was necessary to establish the technical feasibility and impact of the attacks.

Finally, we took care to minimise misuse risk in both experimentation and artefact release. Our evaluation was performed in controlled settings on authorised devices only, and we avoided releasing additional operational details beyond those necessary to support scientific scrutiny and reproducibility. Where mitigation behaviour was observable during the disclosure process, we validated it in our controlled environment and shared those observations privately with Google.

References

[1] Google, "The Google Fast Pair Service," https://developers.google.com/nearby/fast-pair, Accessed 8 Nov. 2025.

[2] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR," in *USENIX Security*, 2019.

[3] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "BIAS: Bluetooth Impersonation AttackS," in *2020 IEEE Symposium on Security and Privacy (S&P)*, 2020.

[4] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, "BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy," in *ASIA CCS*, 2022.

[5] D. Antonioli, "BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23, 2023.

[6] D. Heinze, J. Classen, and F. Rohrbach, "MagicPairing: Apple's take on securing bluetooth peripherals," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20, 2020.

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

[8] L. Böttger, A. Matern, D. Arndt, and M. Hollick, "Okay Google, Where's My Tracker? Security, Privacy, and Performance Evaluation of Google's Find My Device Network," *Proceedings on Privacy Enhancing Technologies*, 2025.

[9] Android Open Source Project, "Android security update severity classification," https://source.android.com/docs/security/overview/updates-resources#severity, Accessed 5 Nov. 2025.

[10] Bluetooth Special Interest Group, "Bluetooth Core Specification Version 5.4," Bluetooth SIG, Tech. Rep., 2023.

[11] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, "SoK: The Long Journey of Exploiting and Defending the Legacy of King Harald Bluetooth," in *2024 IEEE Symposium on Security and Privacy (S&P)*, 2024.

[12] Bluetooth Special Interest Group, "Bluetooth Core Specification Version 4.0," Bluetooth SIG, Tech. Rep., 2010.

[13] B. H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[14] Google, "Provider Advertising Signal," https://developers.google.com/nearby/fast-pair/specifications/service/provider, Accessed 8 Nov. 2025.

[15] Google , "Target Audience," https://developers.google.com/nearby/fast-pair/system-integrator-roles-and-responsibilities, Accessed 9 Nov. 2025.

[16] D. Dolev and A. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[17] T. Jager, J. Schwenk, and J. Somorovsky, "Practical Invalid Curve Attacks on TLS-ECDH," in *ESORICS 2015*, 2015.

[18] A. Antipa, D. Brown, A. Menezes, R. Struik, and S. Vanstone, "Validation of Elliptic Curve Public Keys," in *PKC*, 2003.

[19] E. Biham and L. Neumann, "Breaking the Bluetooth Pairing – The Fixed Coordinate Invalid Curve Attack," in *SAC*, 2019.

[20] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 517–533.

[21] Android, "Explore featured findable tags and devices." https://www.android.com/learn-find-hub/compatible-devices/?device-types=headphones-and-buds, Accessed 7 Nov. 2025.

[22] Google, "Fast Pair Validator," https://play.google.com/store/apps/details?id=com.google.location.nearby.apps.fastpair.validator, Accessed 8 Nov. 2025.

[23] Apple Inc. and Google LLC, "Apple and Google deliver support for unwanted tracking alerts in iOS and Android," https://www.apple.com/newsroom/2024/05/apple-and-google-deliver-support-for-unwanted-tracking-alerts-in-ios-and-android/, press release, May 13 2024.

[24] IETF WG on Detecting Unwanted Location Trackers, "Detecting Unwanted Location Trackers," https://datatracker.ietf.org/doc/charter-ietf-dult/, 2025.

[25] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force (IETF), RFC 8446, 2018.

[26] FIDO Alliance, "Client to Authenticator Protocol (CTAP)," 2019.

[27] W3C, "Web Authentication: An API for accessing Public Key Credentials Level 1," W3C Recommendation, 2019.

[28] J. Wu, P. Traynor, D. Xu, D. J. Tian, and A. Bianchi, "Finding Traceability Attacks in the Bluetooth Low Energy Specification and Its Implementations," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 4499–4516.

[29] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," in *2012 IEEE Symposium on Security and Privacy (S&P)*, 2012.

[30] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," in *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019.

[31] T. Zillner and S. Strobl, "ZigBee Exploited: The Good, the Bad, and the Ugly," in *Black Hat USA*, 2015.

[32] S. Duttagupta, A. Kolozyan, G. Nicolas, B. Preneel, and D. Singelée, "What's the Matter? An In-Depth Security Analysis of the Matter Protocol," Cryptology ePrint Archive, Paper 2025/1268, 2025. [Online]. Available: https://eprint.iacr.org/2025/1268

[33] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method Confusion Attack on Bluetooth Pairing," in *2021 IEEE Symposium on Security and Privacy (S&P)*, 2021.

Appendix A. Security Model and Proofs for *IntentPair*

This appendix formalises *IntentPair* strictly as a *hardening layer* for pairing authorisation. We do *not* model the full security of the Google Fast Pair Service (GFPS), nor claim a complete authenticated key exchange proof for the ecosystem. Instead, we show that *IntentPair* enforces *fail-closed authorisation*: a session is accepted only if (i) explicit pairing intent is present (initial pairing), or (ii) a valid ownership key is demonstrated (re-pairing).

A.1. Protocol Abstraction

We abstract away transport and user-interface behaviour and focus only on the cryptographic checks. Let $\mathbb{G}$ be a group of prime order generated by g. Let T denote the public transcript and κ the desired session key length. Let P denote the Provider with long-term key pair (sk_P, pk_P) and S the Seeker.

Initial Pairing. When pairing mode is active, P samples a volatile intent nonce $I \xleftarrow{\$} \{0,1\}^\lambda$, erased upon exit. The Seeker samples $(x, X = g^x)$ and both compute $z = \mathrm{ECDH}(x, pk_P)$. The session key is derived as:

$$\begin{aligned} \mathsf{prk} &:= \mathrm{HKDF.Extract}(I, z), \\ K &:= \mathrm{HKDF.Expand}(\mathsf{prk}, \text{“IntentPair-init”} \parallel T, \kappa). \end{aligned}$$

Re-pairing. Outside pairing mode, P samples a fresh challenge $c \xleftarrow{\$} \{0,1\}^\lambda$. A legitimate S holding ownership key ak returns $\tau = \mathrm{MAC}_{ak}(c \parallel T)$. If verification succeeds, both derive:

$$\begin{aligned} \sigma &:= H(ak \parallel c), \\ \mathsf{prk} &:= \mathrm{HKDF.Extract}(\sigma, z), \\ K &:= \mathrm{HKDF.Expand}(\mathsf{prk}, \text{“IntentPair-repair”} \parallel T, \kappa). \end{aligned}$$

In both branches, P accepts only if a subsequent AEAD verification under K succeeds.

A.2. Security Model

We use a standard Bellare–Rogaway multi-session model. The adversary $\mathcal{A}$ controls the network via Send queries and may issue CorruptP (revealing sk_P) and CorruptAK (revealing ak).

For an accepted session, the session identifier is

$$sid := (\mathsf{mode}, pk_P, X, T, c)$$

where $\mathsf{mode} \in \{\mathsf{init}, \mathsf{repair}\}$ and $c = \perp$ for initial pairing. Two oracles are partnered if they share the same sid and have opposite roles.

Freshness. An accepted Provider oracle Π_P^i is *fresh* if: (i) no Reveal query was issued to it or its partner, (ii) CorruptP was not issued before acceptance, and (iii) for re-pairing, CorruptAK was not issued before acceptance. Furthermore, for an initial pairing session, freshness requires that P was in pairing mode and thus a valid nonce I was sampled.

Security Goals. The adversary's advantage $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{IntentPair}}$ captures: (i) *authentication*: causing a fresh Provider oracle to accept without a partnered Seeker, and (ii) *key indistinguishability*: distinguishing the session key from random in a Test query against a fresh oracle.

A.3. Security Analysis

Theorem 1. *Let $\mathcal{A}$ be a PPT adversary interacting with at most q sessions. Under the DDH assumption in $\mathbb{G}$, assuming HKDF is a secure PRF, the MAC is SUF-CMA secure, and the AEAD scheme is INT-CTXT secure, we have:*

$$\mathrm{Adv}^{\mathsf{IntentPair}}_{\mathcal{A}} \le \mathrm{Adv}^{\mathsf{DDH}}_{\mathcal{B}_1} + \mathrm{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathcal{B}_2} + \mathrm{Adv}^{\mathsf{PRF}}_{\mathcal{B}_3} + \mathrm{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{B}_4} + \frac{q^2}{2^\lambda}.$$

We additionally assume that the ownership key ak has sufficient min-entropy.

Proof. Let S_i denote the event that $\mathcal{A}$ succeeds in Game i.

Game 0. The real multi-session experiment.

Game 1 (DDH). For sessions where the Seeker's ephemeral key was generated by an honest oracle, replace z with a uniform random string. Then

$$|\Pr[S_0] - \Pr[S_1]| \le \mathrm{Adv}^{\mathsf{DDH}}_{\mathcal{B}_1}.$$

Game 2 (MAC unforgeability). For re-pairing sessions, abort if a valid MAC τ is accepted that was not generated by a legitimate Seeker. Then

$$|\Pr[S_1] - \Pr[S_2]| \le \mathrm{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathcal{B}_2}.$$

Game 3 (HKDF pseudorandomness). Replace (prk, K) with independent random values.

For any accepted initial pairing session, correctness implies that P was in pairing mode and sampled $I \xleftarrow{\$} \{0,1\}^\lambda$ unknown to $\mathcal{A}$. Thus, I contributes λ bits of min-entropy to HKDF.Extract.

For re-pairing, $\sigma = H(ak \parallel c)$ inherits entropy from ak under the min-entropy assumption.

Therefore, even against an active adversary that may know z, distinguishing this game yields an adversary against the PRF security of HKDF:

$$|\Pr[S_2] - \Pr[S_3]| \le \mathrm{Adv}^{\mathsf{PRF}}_{\mathcal{B}_3}.$$

Game 4 (AEAD integrity). Abort if an adversary causes acceptance via a forged ciphertext:

$$|\Pr[S_3] - \Pr[S_4]| \le \mathrm{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{B}_4}.$$

Game 5 (collisions). Collisions in I or c occur with probability at most $q^2/2^\lambda$.

In the final game, authentication and key indistinguishability hold with negligible advantage. Summing the bounds yields the result.

A.4. Discussion: Fail-Closed Authorisation

IntentPair binds authorisation directly into key derivation: acceptance requires knowledge of either the volatile intent nonce I or the ownership key ak. Consequently, bypassing application-layer checks cannot produce a valid session key, enforcing fail-closed behaviour.