IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
AUSTIN DIVISION

SECURE COMMUNICATION TECHNOLOGIES, LLC, Plaintiff, **v.** **GOOGLE LLC,** Defendant.	Case No. 1:25-cv-1207 Jury Trial Demanded

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Secure Communication Technologies, LLC ("SCT") files this Complaint for patent infringement against Defendant Google LLC ("Google") alleging as follows:

BACKGROUND AND NATURE OF THE SUIT

1. This is a claim for patent infringement arising under the patent laws of the United States, Title 35 of the United States Code. This case asserts infringement of United States Patent Nos. 11,334,918 (the "'918 Patent"), 11,443,344 (the "'344 Patent"), and 11,687,971 (the "'971 Patent") (collectively, "the Patents-in-Suit").

2. The Patents-in-Suit are owned by SCT.

3. The inventions described and claimed in the Patents-in-Suit were invented and developed by Mr. James Proctor Jr. (a pioneering researcher, inventor, entrepreneur and well-recognized technical expert in the field of wireless communications) and his father, James A. Proctor III.

THE PARTIES

4. SCT is a Texas limited liability company with its principal place of business at 204 North Fredonia Street, Longview, Texas 75601.

5. Defendant is a limited liability company organized and existing under the laws of Delaware. Google may be served with process through its registered agent, the Corporation Service Company d/b/a/ CSC-Lawyers Inc., at 211 E. 7th Street, Suite 620, Austin, Texas 78701.

JURISDICTION AND VENUE

6. This action arises under the patent laws of the United States, 35 U.S.C. § 101, et seq. This Court's jurisdiction over this action is proper under the above statutes, including 35 U.S.C. § 271, et seq., 28 U.S.C. § 1331 (federal question jurisdiction) and § 1338 (jurisdiction over patent actions).

7. The Court has personal jurisdiction over Defendant consistent with the requirements of the Due Process Clause of the United States Constitution and the Texas Long Arm Statute.

8. Defendant has regularly and systematically transacted business in Texas, directly or through subsidiaries or intermediaries, and/or committed acts of patent infringement in Texas as alleged more particularly below.

9. Defendant has placed infringing products into the stream of commerce by shipping those products into Texas or knowing that the products would be shipped into Texas.

10. In addition, on information and belief, Google conducts business at its Texas facilities related to its mobile devices, including the devices accused of infringement in this case.

11. Venue is proper in this Court pursuant to 28 U.S.C. §§ 1391 and 1400(b).

12. Venue is proper at to Defendant in this district under 28 U.S.C. §1400(b) because Defendant has a regular and established place of business in this district and has committed acts of infringement in this district.

SCT Exhibit 2016
IPR2025-01181

13. On information and belief, Defendant has a regular and established place of business in Austin, Texas, and is responsible for importing and selling smartphones, tablets, laptops, and other mobile devices and equipment in the United States.

14. Defendant makes, uses, sells, offers for sale, and/or imports Google-branded smartphones, tablets, laptops, and other electronics in this District, including the products accused of infringement herein.

15. On information and belief, Defendant has operated, and on information and belief continues to operate, permanent offices located at 500 West Second Street, Austin, Texas, 78701; 601 West Second Street, Austin, Texas 78701; 901 East 5th Street, Austin, Texas 78701; 500 Chichon Street B, Austin, Texas 78702; 3601 South Congress Avenue, J-100, Austin, Texas 78704; 304 East 24th Street, Austin, Texas 78705; 845 Interchange Boulevard, Austin, Texas 78721; 6013 Technical Center Drive, Austin, Texas 78721; 4100 Smith School Road, Austin, Texas 78744; 807 Ruby Drive, Austin, Texas 78753; 10200 McKalla Place, Austin, Texas 78758; 2350 South Midkiff Road, Midland, Texas 79701; and 1100 West County Road, Midland, Texas.

16. Defendant also employs full-time personnel, such as engineers and managers in this district, including in Travis County.

17. Defendant has also committed acts of infringement in this district by commercializing, marketing, selling, distributing, and servicing certain Google-branded devices, including but not limited to phones, tablets, and laptops, which are devices Plaintiff accuses of infringement in this case.

18. Defendant has also committed acts of infringement in this district by developing, commercializing, marketing, selling, and distributing its Android operating system for use in third party phones and tablets, which are devices Plaintiff accuses of infringement in this case.

THE PATENTS-IN-SUIT

The '918 Patent

19. The '918 Patent, entitled "Exchanging identifiers between wireless communication to determine further information to be exchanged or further services to be provided," duly and legally issued on May 17, 2022, from U.S. Patent Application No. 15/271,410, filed on September 21, 2016, naming James A. Proctor, Jr. and James A. Proctor III as the inventors. A true and correct copy of the '918 Patent is attached hereto as Exhibit 1 and is incorporated by reference.

20. The '918 Patent claims priority to U.S. Patent Application No. 14/861,563, which was filed on September 22, 2015. The '918 Patent also claims priority to U.S. Patent Application Nos. 14/472,477 (filed on August 29, 2014 and which issued as U.S. Patent No. 9,161,164), 13/775,435 (filed on February 25, 2013 and which issued as U.S. Patent No. 8,849,698), 13/212,723 (filed on August 18, 2011 and which issued as U.S. Patent No. 8,385,896), and 12/364,828 (filed on February 3, 2009 and which issued as U.S. Patent No. 8,090,359). The '918 Patent also claims priority to U.S. Provisional Patent Application Nos. 61/095,359 (filed on September 9, 2008) and 61/095,001 (filed on September 8, 2008).

21. The '918 Patent claims patent-eligible subject matter under 35 U.S.C. § 101.

22. The '918 Patent claims are not directed to an abstract idea and are thus patent-eligible under 35 U.S.C. § 101.

23. The '918 Patent claims contain inventive concepts such that they are patent-eligible under 35 U.S.C. § 101.

24. SCT is the owner and assignee of all rights, title, and interest in and under the '918 Patent.

25. SCT has standing to sue for infringement of the '918 Patent.

The ’344 Patent

26. The ’344 Patent, entitled “Efficient and secure communication using wireless service identifiers,” duly and legally issued on September 13, 2022, from U.S. Patent Application No. 17/366,826, filed on July 2, 2021, naming James A. Proctor, Jr. and James A. Proctor III as the inventors. A true and correct copy of the ’344 Patent is attached hereto as Exhibit 2 and is incorporated by reference.

27. The ’344 Patent claims priority to U.S. Patent Application No. 16/817,896, which was filed on March 13, 2020 and issued as U.S. Patent No. 11,074,615. The ’344 Patent also claims priority to U.S. Patent Application Nos. 15/271,410 (filed on September 21, 2016), 14/861,563 (filed on September 22, 2015), 14/472,477 (filed on August 29, 2014 and which issued as U.S. Patent No. 9,161,164), 13/775,435 (filed on February 25, 2013 and which issued as U.S. Patent No. 8,849,698), 13/212,723 (filed on August 18, 2011 and which issued as U.S. Patent No. 8,385,896), and 12/364,828 (filed on February 3, 2009 and which issued as U.S. Patent No. 8,090,359). The ’344 Patent also claims priority to U.S. Provisional Patent Application Nos. 61/095,359 (filed on September 9, 2008) and 61/095,001 (filed on September 8, 2008).

28. The ’344 Patent claims patent-eligible subject matter under 35 U.S.C. § 101.

29. The ’344 Patent claims are not directed to an abstract idea and are thus patent-eligible under 35 U.S.C. § 101.

30. The ’344 Patent claims contain inventive concepts such that they are patent-eligible under 35 U.S.C. § 101.

31. SCT is the owner and assignee of all rights, title, and interest in and under the ’344 Patent.

32. SCT has standing to sue for infringement of the ’344 Patent.

SCT Exhibit 2016
IPR2025-01181

The ’971 Patent

33. The ’971 Patent, entitled “Efficient and secure communication using wireless service identifiers,” duly and legally issued on June 27, 2023, from U.S. Patent Application No. 17/942,197, filed on September 12, 2022, naming James A. Proctor, Jr. and James A. Proctor III as the inventors. A true and correct copy of the ’971 Patent is attached hereto as Exhibit 3 and is incorporated by reference.

34. The ’971 Patent claims priority to U.S. Patent Application Nos. 17/366,826 (filed on June 2, 2021 and which issued as the ’344 Patent), 16/817,896 (filed on March 13, 2020 and which issued as U.S. Patent No. 11,074,615), 15/271,410 (filed on September 21, 2016), 14/861,563 (filed on September 22, 2015), 14/472,477 (filed on August 29, 2014 and which issued as U.S. Patent No. 9,161,164), 13/775,435 (filed on February 25, 2013 and which issued as U.S. Patent No. 8,849,698), 13/212,723 (filed on August 18, 2011 and which issued as U.S. Patent No. 8,385,896), and 12/364,828 (filed on February 3, 2009 and which issued as U.S. Patent No. 8,090,359). The ’344 Patent also claims priority to U.S. Provisional Patent Application Nos. 61/095,359 (filed on September 9, 2008) and 61/095,001 (filed on September 8, 2008).

35. The ’971 Patent claims patent-eligible subject matter under 35 U.S.C. § 101.

36. The ’971 Patent claims are not directed to an abstract idea and are thus patent-eligible under 35 U.S.C. § 101.

37. The ’971 Patent claims contain inventive concepts such that they are patent-eligible under 35 U.S.C. § 101.

38. SCT is the owner and assignee of all rights, title, and interest in and under the ’971 Patent.

39. SCT has standing to sue for infringement of the ’971 Patent.

GENERAL ALLEGATIONS OF PATENT INFRINGEMENT

40. As detailed below, Defendant has infringed (and continues to infringe) one or more claims of each Patent-in-Suit by making, using, selling, offering for sale, and/or importing into the United States, and/or practicing the claimed method via its use of, certain Google-branded phones and tablets, including but not limited to the Pixel 9a, Pixel 9 Pro, Pixel 9 Pro XL, Pixel 9 Pro Fold, Pixel 9, Pixel 8 Pro, Pixel 8, Pixel 8a, and Pixel Tablet.

41. Also, as detailed below, Defendant has infringed (and continues to infringe) one or more claims of each Patent-in-Suit through the making, using, selling, offering for sale, and/or importing into the United States, and/or practicing the claimed method via its use of, certain Chromebook laptops, including those branded, marketed, and sold by Acer, ASUS, HP, and Lenovo.

42. Also, as detailed below, Defendant has infringed (and continues to infringe) one or more claims of each Patent-in-Suit through the making, using, selling, offering for sale, and/or importing into the United States, and/or practicing the claimed method via its use of, certain Android smartphones, including those branded, marketed, and sold by Asus, Motorola Mobility and OnePlus.

43. The products described and identified in preceding paragraphs numbered 40 through 42 are referred to herein as the "Accused Products."

44. The Accused Products infringe one or more claims of each Patent-in-Suit via several features and associated functionalities including, but not limited to, Google's Nearby Share/Quick Share, Fast Pair, and Find Hub (previously known as Find My) features.

45. On information and belief, Defendant has had knowledge of the Patents-in-Suit and of its infringement of the Patents-in-Suit at least through receipt or knowledge of SCT's Original

Complaint in *Secure Communication Technologies, LLC v. Samsung Electronics Co., Ltd. et al.*, No. 2:24-cv-00484-JRG (the "Samsung Case"), in the U.S. District Court for the Eastern District of Texas and/or through receipt or knowledge of SCT's Infringement Contentions in the Samsung Case, in which SCT identified how the Accused Products in that case (including products that utilize Google's Android operating system) infringe claims of the Patents-in-Suit. Alternatively, Defendant has had knowledge of the Patents-in-Suit, and its infringement of the Patents-in-Suit, at least since sometime before July 2, 2025, when Defendant filed IPR Nos. 2025-01181, -1182, and -1183, challenging the validity of the Patents-in-Suit and referencing the Samsung Case.

46. Defendant was aware of Mr. Jim Proctor, the co-inventor of the Patents-in-Suit, prior to the filing of this lawsuit via Mr. Proctor's expert work and his prior dealings with Google.

47. Defendant's acts of infringement have caused damage to SCT. SCT is entitled to recover from Defendant the damages sustained by SCT as a result of Defendant's wrongful acts in an amount subject to proof at trial.

48. Further discovery may reveal earlier knowledge of one or more of the Patents-in-Suit, which would provide additional evidence of Defendant's willful infringement of the Patents-in-Suit.

49. For each count of infringement listed below, SCT incorporates and re-states the allegations contained in the preceding paragraphs above, including these General Allegations, as if fully set forth in each count of infringement.

COUNT I – INFRINGEMENT OF THE '918 PATENT

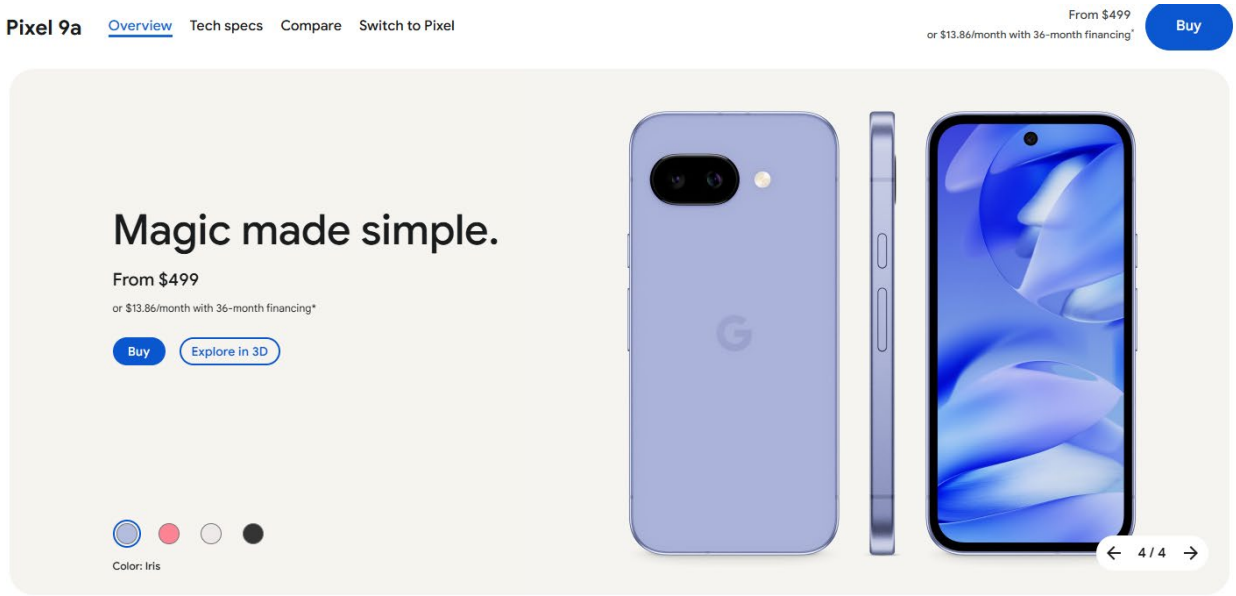
50. SCT incorporates herein the allegations made in paragraphs 1 through 49.

51. Defendant has directly infringed one or more claims of the '918 Patent, including, for example, claims 1 and 9, in violation of 35 U.S.C. § 271(a) by making, using, selling, offering

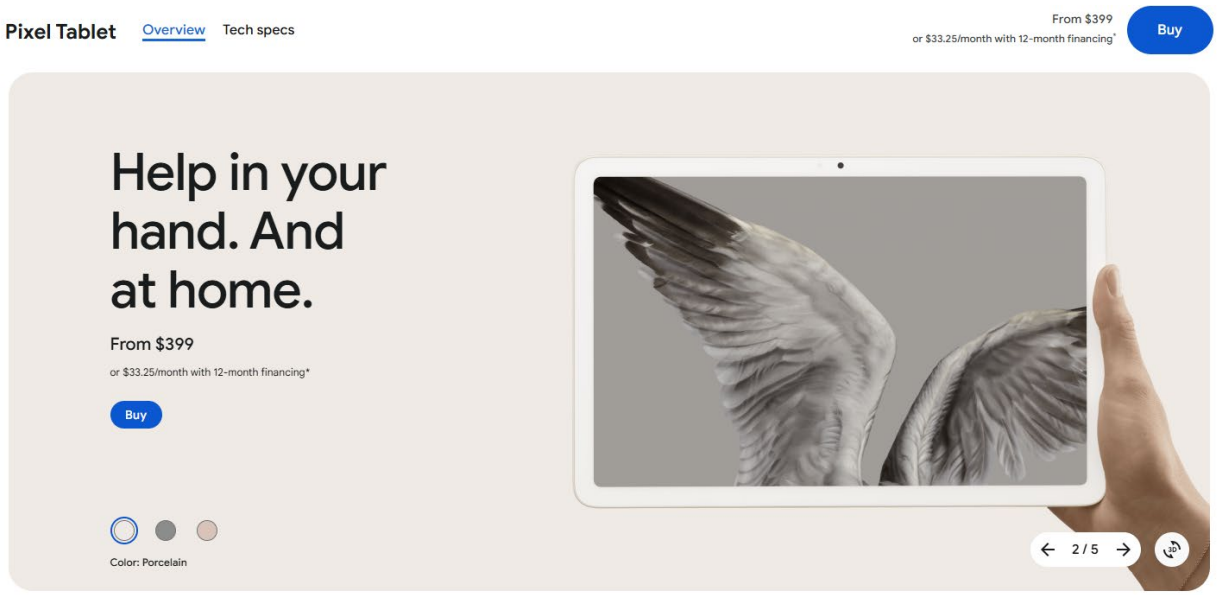
for sale, and/or importing into the United States infringing products including, but not limited to, the Accused Products.

52. Additionally, Defendant has indirectly infringed the ’918 Patent in violation of 35 U.S.C. § 271(b) at least by inducing customers to purchase the Accused Products and/or by instructing customers how to use the Accused Products in a way that directly infringes at least claims 1 and 9 of the ’918 Patent.

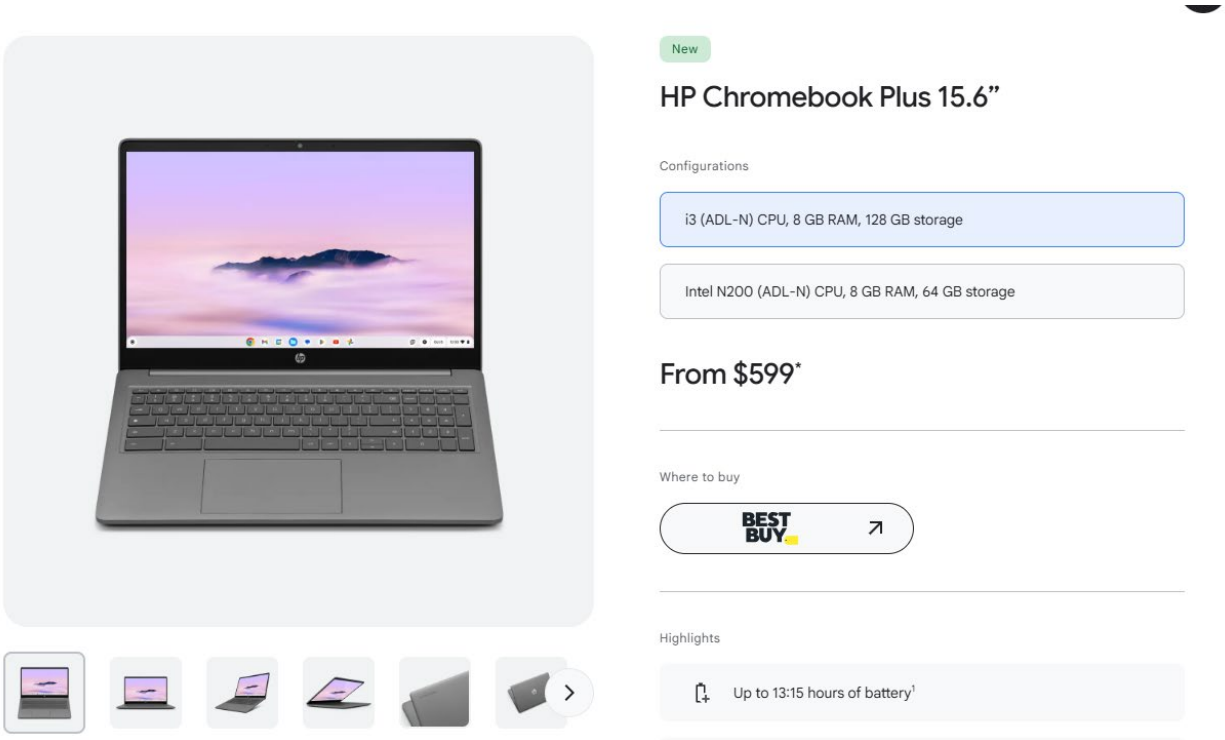
53. Each of the Accused Products is “[a] mobile wireless device.”



https://store.google.com/product/pixel_9a?hl=en-US



https://store.google.com/product/pixel_tablet?hl=en-US



https://www.google.com/chromebook/shop-chromebooks/?modal-id=hp-chromebook-156-inch-pdp-modal

https://www.motorola.com/us/en/p/phones/razr/razr-plus-gen-3/razrplus2025?pn=PB2J0018US&tab=techspecs

https://www.oneplus.com/us/oneplus-13

54. Each of the Accused Products includes "a first radio" (*e.g.*, radio(s) that operate in accordance with cellular data (LTE, 5G), WiFi wireless local area network, or Bluetooth (BT) standards).

Connectivity and Location	Wi-Fi 6E with 2.4GHz+5GHz+6GHz, 2x2 MIMO Bluetooth® v5.3 with antenna diversity for enhanced quality and connection NFC Google Cast Dual Band GNSS GPS, GLONASS, Galileo

https://store.google.com/product/pixel_9a_specs?hl=en-US

Connectivity and location

Wi-Fi 6 (802.11 a/b/g/n/ac/ax) with 2x2 MIMO, simultaneous dual-band (2.4 GHz, 5.0 GHz)

Bluetooth v5.2

Ultra-Wideband chip for accurate ranging

Google Cast

https://store.google.com/product/pixel_tablet_specs?hl=en-US

Wireless Connectivity	Bluetooth, Wi-Fi
Wireless Networking Standard	Wi-Fi 6

https://www.bestbuy.com/site/hp-15-6-full-hd-chromebook-plus-laptop-with-google-ai-intel-core-i3-8gb-memory-128gb-ufs-mineral-silver/6553105.p?skuId=6553105

55. Each of the Accused Products includes "a second radio" (*e.g.*, radio(s) that operate in accordance with cellular data (LTE, 5G), WiFi wireless local area network or a Bluetooth (BT) and/or a Bluetooth LE (low energy) (BLE) radio standard)).

Connectivity and Location

Wi-Fi 6E with 2.4GHz+5GHz+6GHz, 2x2 MIMO

Bluetooth® v5.3 with antenna diversity for enhanced quality and connection

NFC

Google Cast

Dual Band GNSS
GPS, GLONASS, Galileo

https://store.google.com/product/pixel_9a_specs?hl=en-US

Connectivity and location

Wi-Fi 6 (802.11 a/b/g/n/ac/ax) with 2x2 MIMO, simultaneous dual-band (2.4 GHz, 5.0 GHz)

Bluetooth v5.2

Ultra-Wideband chip for accurate ranging

Google Cast

https://store.google.com/product/pixel_tablet_specs?hl=en-US

Wireless Connectivity ⓘ	Bluetooth, Wi-Fi
Wireless Networking Standard ⓘ	Wi-Fi 6

https://www.bestbuy.com/site/hp-15-6-full-hd-chromebook-plus-laptop-with-google-ai-intel-core-i3-8gb-memory-128gb-ufs-mineral-silver/6553105.p?skuId=6553105

Connectivity

Headphone Jack	No
Wireless Connectivity	Bluetooth, Wi-Fi, NFC
Wireless Networking Standard	Wi-Fi 6
DisplayPort Over USB-C	None

https://www.bestbuy.com/site/motorola-razr-2025-256gb-unlocked-pantone-mocha-mousse/6625237.p?skuId=6625237

Connectivity

LTE/LTE-A

4×4 MIMO, Supports up to DL Cat 20/UL Cat 18 (2000Mbps /200Mbps), depending on carrier support

Band

2G GSM: 850/900/1800/1900MHz
3G WCDMA:Bands 1/2/4/5/6/8/19
4G LTE FDD: Bands 1/2/3/4/5/7/8/12/13/17/18/19/20/25/26/28/30/32/66/71
4G LTE TDD: Bands 38/39/40/41/48
5G SA: n1/n2/n3/n5/n7/n8/n12/n20/n25/n28/n30/n38/n40/n41/n48/n66/n71/n75/n77/n78

Wi-Fi

Wi-Fi 7, Wi-Fi 6, Wi-Fi 5, WLAN Display, WLAN tethering, WLAN overlay, 160 MHz Wi-Fi channels over 5 GHz, 320 MHz Wi-Fi channels over 6 GHz, 2X2 MIMO

Bluetooth

Bluetooth® 5.4
SBC, AAC, aptx, aptx-HD, LDAC, LHDC5.0

NFC

NFC enabled

Positioning

GPS(L1+L5), Galileo(E1+E5a)

https://www.oneplus.com/us/oneplus-13

56. Each of the Accused Products includes "one or more processors."

Processor	Google Tensor G4 Titan M2 security coprocessor

https://store.google.com/product/pixel_9a_specs?hl=en-US

Processor

Google Tensor G2

Titan M2 security coprocessor

https://store.google.com/product/pixel_tablet_specs?hl=en-US

Processor Model Intel Core i3

https://www.bestbuy.com/site/hp-15-6-full-hd-chromebook-plus-laptop-with-google-ai-intel-core-i3-8gb-memory-128gb-ufs-mineral-silver/6553105.p?skuId=6553105

performance

Internal Storage
256GB[8]
UFS 4.0

Memory (RAM)
12GB
LPDDR5X

Operating System
Android™ 15

Processor
Snapdragon 8s Gen 3 Mobile Platform

Security
Side fingerprint reader, Face unlock

Sensors
Fingerprint reader, Proximity + light sensor, Ambient light sensor, Accelerometer, Gyroscope, eCompass, Hall sensor, Barometer

https://www.motorola.com/us/en/p/phones/razr/razr-plus-gen-3/razrplus2025?pn=PB2J0018US&tab=techspecs

Specs

Key Features

Platform

Snapdragon® 8 Elite Mobile Platform

https://www.oneplus.com/us/oneplus-13

57. Each of the Accused Products includes "one or more memory devices coupled to the one or more processors, the one or more memory devices storing a set of instructions that when

executed by the one or more processors cause the one or more processors to perform operations," including the claimed operations as set forth herein in the following paragraphs.

Memory and Storage	8 GB RAM 128 GB / 256 GB[10]

https://store.google.com/product/pixel_9a_specs?hl=en-US

Memory and storage

8 GB LPDDR5 RAM

128 & 256 GB UFS 3.1 storage[4]

https://store.google.com/product/pixel_tablet_specs?hl=en-US

Total Storage Capacity	128 gigabytes
System Memory (RAM)	8 gigabytes

https://www.bestbuy.com/site/hp-15-6-full-hd-chromebook-plus-laptop-with-google-ai-intel-core-i3-8gb-memory-128gb-ufs-mineral-silver/6553105.p?skuId=6553105

performance			
	Internal Storage 256GB[8] UFS 4.0	**Memory (RAM)** 12GB LPDDR5X	**Operating System** Android™ 15
	Processor Snapdragon 8s Gen 3 Mobile Platform	**Security** Side fingerprint reader, Face unlock	**Sensors** Fingerprint reader, Proximity + light sensor, Ambient light sensor, Accelerometer, Gyroscope, eCompass, Hall sensor, Barometer

https://www.motorola.com/us/en/p/phones/razr/razr-plus-gen-3/razrplus2025?pn=PB2J0018US&tab=techspecs

Performance

Performance

Operating System: OxygenOS 15.0 based on Android™ 15
Platform: Snapdragon® 8 Elite Mobile Platform
CPU: Qualcomm® Oryon™ CPU @4.32GHz
GPU: Adreno™ 830
RAM: 12GB/16GB LPDDR5X
Storage: 256GB/512GB UFS 4.0
Battery: 6,000 mAh (Dual-cell 3,000 mAh, non-removable)
Vibration: Haptic motor
Available configurations: 12GB+256GB / 16GB+512GB

https://www.oneplus.com/us/oneplus-13

58. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, using the first radio, identifier related information associated with a proximity beacon service, from one or more servers." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by receiving, using a first radio, identifier related information associated with the particular service—*e.g.*, Nearby Share/Quick Share, Fast Pair, Find Hub—from one or more servers.

59. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

A new way to share files

Sharing pictures, files and text between devices should be simple, fast and convenient. In 2020, we introduced Nearby Share to make it easy to share files across devices. Samsung also has its own capability called Quick Share, which is loved by their users.

Collaborating with Samsung, we're bringing the best of our sharing solutions together into a singular cross-Android solution under the Quick Share [1] name. We've integrated the experiences and created the best default, built-in option for peer-to-peer content sharing across all types of devices in the Android and Chromebook ecosystems. And to make sharing even more seamless between devices, we're working with leading PC manufacturers like LG to expand Quick Share to Windows PCs as a pre-installed app.

This means with a simple tap of the new Quick Share icon, you can see a list of available devices close by. You remain in control of your privacy, and can choose in your phone's settings who can discover your device and send files, whether it's everyone, only your contacts or just your own devices. Quick Share will start rolling out to current Nearby Share enabled devices next month.

https://blog.google/products/android/ces-2024-android-updates/

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

With Quick Share, you can immediately send and receive files from devices close to your location.

Important:

- Quick Share is available on Android 6+ devices and Chromebooks, and on selected Windows devices through an App.
- For a Galaxy device with Android 10 & One UI 2.1 or later, the settings and features can be different. Learn about Quick Share feature on Galaxy devices .
- This feature was previously known as Nearby Share. If you still find Nearby Share on some devices, Quick Share will still work.

https://support.google.com/android/answer/9286773

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share .
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code** .
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

https://support.google.com/android/answer/9286773

Adjust Quick Share settings

1. At the top of your screen, swipe down.
2. Tap Quick Share .
   - If there's no Quick Share :
     a. Tap Edit .
     b. Under "Quick Settings," add Quick Share .
3. Choose who can share with you:
   - **Your devices:** If your screen is off, your device is still visible to your other devices with the same Google Account.
   - **Contacts:** While your screen is on and unlocked, your device is visible to your nearby contacts.
   - **Everyone:** While your screen is on and unlocked, your device is visible to anyone nearby.
     - **Only for 10 minutes:** To protect your privacy, your visibility automatically switches back to your previous selection after 10 minutes.
4. To manage additional settings, like how you appear to others, at the lower left, tap **Settings**.

https://support.google.com/android/answer/9286773

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

Back up & sync device & SIM contacts

To back up and sync your device and SIM contacts, save them as Google contacts:

1. On your Android phone or tablet, open the "Settings" app.
2. Tap **Google** › **Settings for Google apps** › **Google Contacts sync** › **Also sync device and SIM contacts** › **Sync device and SIM contacts automatically.**
3. Turn on **Sync device and SIM contacts automatically.**
4. Pick the account you'd like your contacts to be saved in. Your contacts can only be automatically saved to one Google Account.

https://support.google.com/contacts/answer/9423168?hl=en

60. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account -- you can rename devices to whatever you want. Whenever the service is able to detect an accessory near other known host devices, it'll automatically prompt the user to pair with the accessory.

GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account.

GFPS can also display your accessory's battery levels in a persistent notification, aid in passing off audio sourcing (if your audio gear supports multipoint) to a different device as needed, provide an interface on your phone to control certain features on your earbuds like ANC, and helps you locate your accessory by either ringing an alarm through them or showing their last known location while paired.

https://www.pocket-lint.com/what-is-google-fast-pair-and-how-does-it-work/

Characteristic: Key-based Pairing

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

This characteristic controls the Key-based Pairing procedure. In this procedure, a certain level of trust is established by verifying that the Seeker and Provider are both in possession of a pre-shared key. The key is different in each case:

- Case 1: The pre-shared key is based on the anti-spoofing public/private key pair, and the Seeker's own public/private key pair which will change for each pairing attempt.
  - The Provider is in pairing mode.
  - The Seeker verifies that the Provider is in possession of the anti-spoofing private key.

  Note that when in pairing mode, the Provider may also of course pair in the usual way, for example, to pair with a device that does not support Fast Pair's Key-based Pairing.
- Case 2: The pre-shared key is one of the account keys.
  - The Provider is usually not in pairing mode. (But this is not a requirement—The Provider should support using an account key even when in pairing mode.)
  - The Seeker and Provider each verify that the other is in possession of the account key.

Since both cases are extremely similar, except for which pre-shared key is used, they are combined in procedure.

https://developers.google.com/nearby/fast-pair/specifications/characteristics#KeyBasedPairing

61. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

62. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, using the second radio, a plurality of short range transmissions, including a plurality of proximity beacon transmissions when the device is located within a detection range of one or more beacon transmitter devices associated with the proximity beacon service, each of the proximity beacon transmissions including a respective MAC address, a respective unique identifier, and a Proximity Beacon Service Identifier (PBSI)." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by receiving, using the second radio (*e.g.*, a Bluetooth Low Energy radio), a plurality of short range transmissions, including a plurality of proximity beacon transmissions when the device is located within a detection range of one or more beacon transmitter devices associated with the proximity beacon service. Each of the proximity beacon transmissions includes a respective MAC address, a respective unique identifier, and a Proximity Beacon Service Identifier (PBSI).

63. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

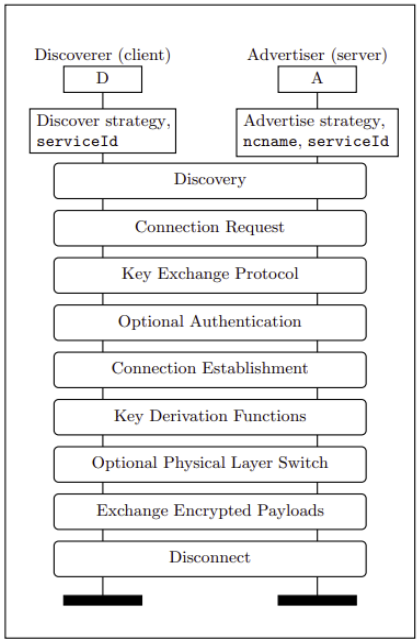


Fig. 1: The Nearby Connections API has two types of actors: the Discoverer (client) and the Advertiser (server). It uses application layer encryption and optional user authentication.

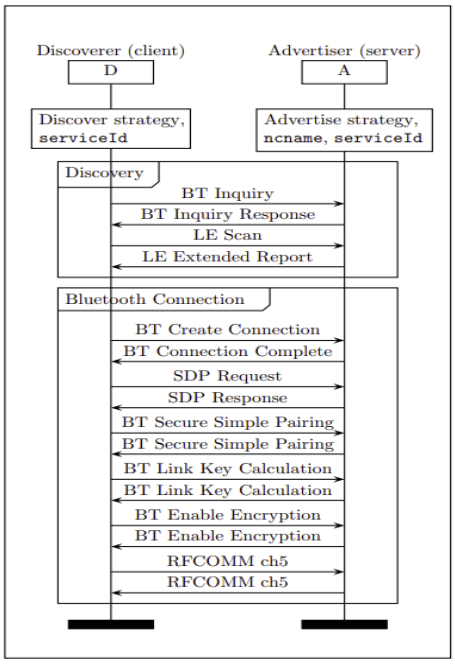


Fig. 3: Nearby Connections Request. BT is Bluetooth BR/EDR. Secure Simple Pairing provides link-layer encryption.

D Antononioli, N.O. Tippenhauer and K. Rasmussen, "Nearby threats: reversing, analyzing and attacking Google's Nearby Connections on Android," Network and Distribution Systems Security (NDSS) Symposium, 2019, San Diego CA.

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

Public Methods

public DiscoveryOptions **build** ()

public DiscoveryOptions.Builder **setLowPower** (boolean lowPower)

Sets whether to use low power. If true, only low power mediums (like BLE) will be used for discovery. By default, this option is false.

public DiscoveryOptions.Builder **setStrategy** (Strategy strategy)

Sets the discovery strategy. Must match the strategy used in `AdvertisingOptions`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/DiscoveryOptions.Builder

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

A. Discovery and Connection Request

A nearby connection is always requested using Bluetooth, regardless of the nearby connection parameters. Discovering and advertising are done in a deterministic and predictable way without using encryption. An attacker who posses a clone of the library can pretend to be any advertiser and discoverer of any application, and he can use any Bluetooth compatible device to request a connection. The Bluetooth connection uses Secure Simple Pairing (SSP) with a link key that is not authenticated and not persistent. Indeed, an attacker could insert himself as a man in the middle in the Bluetooth link and he can force the re-establishment of the link key at any time.

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

https://francozappa.github.io/publication/rearby/paper.pdf

As in the connection request phase, Wi-Fi and Bluetooth LE are not used to establish a nearby connection. We discovered that devices with the same `ncname` are allowed to connect, this means that in a nearby connection only the `endpointId` uniquely identifies a node. While testing the connection establishment phase we discovered interesting things about the `serviceId`. Any advertiser leaks its `serviceId` if queried with a generic SDP request, e.g., using `sdptool browse adv_btaddress`. Moreover, two devices from different applications can use the same `serviceId` to establish a connection. This means that it is possible to predict the `serviceId` of any application . We also discovered that the library is still using an undocumented `serviceId` named `__LEGACY_SERVICE_ID__`.

https://francozappa.github.io/publication/rearby/paper.pdf

EndpointDiscoveryCallback

On this page

Public Constructor Summary
Public Method Summary
Inherited Method Summary
Public Constructors
Public Methods

public abstract class **EndpointDiscoveryCallback** extends Object

Listener invoked during endpoint discovery.

Public Constructor Summary

EndpointDiscoveryCallback()	

Public Method Summary

abstract void	onEndpointFound(String endpointId, DiscoveredEndpointInfo info) Called when a remote endpoint is discovered.
abstract void	onEndpointLost(String endpointId) Called when a remote endpoint is no longer discoverable; only called for endpoints that previously had been passed to `onEndpointFound(String, DiscoveredEndpointInfo)`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/EndpointDiscoveryCallback

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

A. *Discovery and Connection Request*

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

Advertise and discover

Once the user has granted all required permissions, your app can begin to advertise and discover in order to find nearby devices.

First, choose a `Strategy` for your use case. The `Strategy` you select determines the connection topology for your app (one advertiser to N discoverers, or M advertisers to N discoverers).

On devices that will advertise, call `startAdvertising()` with the desired `Strategy` and a `serviceId` parameter that identifies your app.

On devices that will discover nearby advertisers, call `startDiscovery()` with the same `Strategy` and `serviceId`.

The `serviceId` value must uniquely identify your app. As a best practice, use the package name of your app (for example, `com.google.example.myapp`).

The following example shows how to advertise:

```
private void startAdvertising() {
  AdvertisingOptions advertisingOptions =
      new AdvertisingOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startAdvertising(
          getLocalUserName(), SERVICE_ID, connectionLifecycleCallback, advertisingOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're advertising!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We were unable to start advertising.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

64. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Configuration

Roles

The profile defines two roles: *Fast Pair Seeker*, and *Fast Pair Provider*. The Seeker is normally a phone, looking for a device to pair with. The Provider is a device that is advertising its presence and readiness to pair (e.g. a discoverable pair of headphones).

The Fast Pair Seeker shall use the GAP Central role. The Fast Pair Provider shall use the GAP Peripheral role.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Fast Pair uses Bluetooth Low Energy (BLE) technology to communicate with your phone. BLE is a wireless protocol that consumes very little power and has a long range. It also has a high level of security, as it encrypts the data that is transmitted between devices. Fast Pair uses BLE to send a unique identifier from your device to your phone, which then contacts Google's servers to get more information about the device, such as its name and icon. This way, you can easily recognize your device on your phone screen and pair it with a single tap.

https://communityin.oppo.com/thread/1301071443236225027

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/introduction

Model Registration

All Provider models must be registered with Google before they will work with Fast Pair. After registration, Google will distribute a Model ID and Anti-Spoofing Public/Private Key Pair. The information provided during registration is used in the pairing suggestion presented to the user, and may be used in other UX.

Example pairing suggestion (subject to change)

https://developers.google.com/nearby/fast-pair/specifications/service/modelregistration

Provider Advertising signal

Advertising: When discoverable

When the Provider device is BR/EDR discoverable (that is, in pairing mode), it shall advertise Fast Pair Model ID Data over BLE, and the BLE address shall not be rotated.

Advertising interval: When discoverable

The interval between advertisements should be no larger than 100ms (10Hz). A fast rate allows the Seeker to quickly find the Provider, even when scanning in low-power mode.

Advertising payload: Fast Pair Model ID Data

The advertisement shall contain the Service Data data type, ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0-2	`uint24`	24-bit model ID	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Advertising: When not discoverable

When not discoverable (that is, not in pairing mode), the Provider device shall advertise Fast Pair Account Data, using the following guidelines.

Advertising the account data allows Seekers nearby to recognize when a provider belongs to their account and initiate pairing without having to force the provider back into pairing mode first, which is a common cause for user complaint. Seekers will provide the opportunity for users to be able to ignore this broadcast in the case where they do not wait to pair with the provider or the broadcast is not relevant (for example, if they have already paired). Seekers will also filter out obviously bad broadcasts automatically, such as when the account data is misconfigured.

Advertising interval: When not discoverable

The interval between advertisements should be at most 250ms (4Hz).

Advertising payload: Fast Pair Account Data

The advertisement shall contain the Service Data data type, Ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0	`uint8`	Version and flags 0bVVVVFFFF • V = version • F = flags	`0x00` (reserved for future use)
1 - *varies*		Account Key Data	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

BLE address information

To prevent tracking, BLE advertising shall use the random resolvable private address (*RPA*). The address shall be rotated at minimum every 15 minutes while the device is actively advertising, and every time the state changes from not advertising to advertising. A randomized offset should be used to alter the address randomization interval.

https://developers.google.com/nearby/fast-pair/specifications/configuration

The Account Key Data contains:

Octet	Data type	Description	Value
0	`uint8`	Field length and type 0bLLLLTTTT • L = length of account key filter in bytes • T = type	0bLLLL0000 • length = 0bLLLL = *varies* • type = 0b0000 (show UI indication) or 0b0010 (hide UI indication), Account Key Filter
1 - *s*		Account Key Filter	*varies*
s + 1	`uint8`	Field length and type 0bLLLLTTTT • L = length in bytes • T = type	0b00100001 • length = 0b0010 = 2 • type = 0b0001, Salt
s + 2 - *s* + 3	`uint16`	Salt	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Salt field

The salt is a random value that is appended to account keys when building the bloom filter. This salt should be regenerated every time the RPA is updated for the Provider to avoid tracking across address rotation.

To generate the Account Key Filter using the salt:

1. Generate a random 2-byte *S*. Note that there is no "endianness" to this value, since there is no more or less significant byte — don't alter the byte order.
2. Use the 2-byte *S* as the Salt.
3. In the advertised Fast Pair Account Data, include the generated filter in the Account Key Filter field, and *S* in the Salt field.

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Keys: Account Key List

The Provider shall allocate space to store a persisted list of 128-bit Account Keys. Each Account Key allows the Provider to be recognized as belonging to a certain user account.

The list must be able to store at least five keys (that is, there must be at least 80 bytes of space dedicated to this list). Providers can optionally store more than this, they just must make sure that the keys will fit inside of their advertising packet. The exact number that can be stored will depend on how many free bytes are available in the advertising packet; see the Account Key Filter section for more information on determining how many bytes each key will take up. For example, to advertise 10 account keys, 15 bytes need to be available in the packet.But for personal devices (e.g. headphones), the number of account keys should not be greater than 5. This is to avoid the number of account keys becoming too large, and hence it could be unique and trackable.

This list is initially empty, and must be cleared if the Provider is factory-reset (if the user clears its paired device list). The list is populated as described in the Account Key characteristic section.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Account Key Filter

★ **Note:** Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

The Account Key Filter is a variable-length Bloom filter constructed as follows:

1. Let *n* be the number of account keys (*n* >= 1) in the persisted Account Key list.
2. Let *s*, the size of the filter in bytes, be (1.2**n* + 3) truncated. For example, if 1 key is persisted, *s* = 4 bytes.
   `uint8_t s = (((uint8_t)(( float )1.2 * n)) + 3);`
3. Initialize the filter *F* as an array of *s* bytes, each set to 0.
   `uint8_t F[s] = {0};`
4. For each account key *K* in the persisted Account Key list:
   a. Let *V* be concat(*K*, Salt).

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

```
// In the sample code, the size of salt is 2 bytes.
#define SALT_SIZE 2

uint8_t V[FASTPAIR_ACCOUNT_KEY_SIZE + SALT_SIZE];
for (uint8_t keyIndex = 0; keyIndex < n; keyIndex++)
  {
     // concat (K, Salt)
      fastpair_get_account_key_by_index(keyIndex, V);

      uint8_t randomSalt = (uint8_t)rand();
      V[FASTPAIR_ACCOUNT_KEY_SIZE] = randomSalt;
      ... }
```

b. Hash *V* using SHA256, obtaining a 32-byte value $H = \{H_0, ..., H_{31}\}$.

```
uint8_t H[32] = {0};
SHA256_hash_function(V, H);
```

c. Divide *H* into eight 4-byte unsigned integers in big-endian, $X = \{X_0, ..., X_7\}$, where $X_0 = 0xH_0H_1H_2H_3$.

```
     uint32_t X[8];
     for (index = 0; index < 8; index++)
     {
        X[index] = (((uint32_t)(H[index * 4])) << 24) |
                    (((uint32_t)(H[index * 4 + 1])) << 16) |
                    (((uint32_t)(H[index * 4 + 2])) << 8) |
                    (((uint32_t)(H[index * 4 + 3])) << 0);
     }
```

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

Characteristics

Fast Pair Service

The Fast Pair Provider shall have the following GATT service.

Service	UUID
Fast Pair Service	`0xFE2C`

This service shall have the following characteristics.

Fast Pair Service characteristic	Encrypted	Permissions	UUID
Model ID	No	Read	`FE2C1233-8366-4814-8EB0-01DE32100BEA`
Key-based Pairing	No	Write and notify	`FE2C1234-8366-4814-8EB0-01DE32100BEA`
Passkey	No	Write and notify	`FE2C1235-8366-4814-8EB0-01DE32100BEA`
Account Key	No	Write	`FE2C1236-8366-4814-8EB0-01DE32100BEA`

https://developers.google.com/nearby/fast-pair/specifications/characteristics

Besides the seamless connections, Fast Pair also stores details about the Bluetooth accessory on to your Google ID, so every device running that Google account can automatically identify the accessory. After being added to your Google account, the accessory also appears among other devices in Google's Find My Device app, which can be used to track the accessory's whereabouts or details of the last Android device connected in case you lose it.

https://www.slashgear.com/1454224/google-fast-pair-service-android-how-to/. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

1.1 Definition

- **Initial pairing** is the sequence of events that occur when a user pairs a device to a Google Account signed-in on phone for the very first time. In this sequence, a phone detects the advertisement from the device and displays a notification prompting the user to connect to and save the device. (In this guideline, 'device' means the Bluetooth headset or speaker instead of a reference phone).
- **Subsequent pairing** is the sequence of events that occur when a user signs into their Google Account on a new phone and attempts to pair a device already saved to their Google Account. In this sequence, the new phone recognizes that the advertised Model ID is already saved to the user's Google Account and provides a notification to expedite pairing the device to this phone.

https://developers.google.com/nearby/fast-pair/fast-pair-certification-guideline

65. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page
GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

> ★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

66. Each of the Accused Products is capable of performing, and does perform, the operation of "determining if a particular short range transmission of the plurality of short range transmissions a) includes a MAC address, and b) includes the proximity beacon service identifier (PBSI) and if c) the PBSI indicates the particular short range transmission is a proximity beacon

transmission associated with the proximity beacon service." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by determining if a particular short range transmission of the plurality of short range transmissions a) includes a MAC address, and b) includes the proximity beacon service identifier (PBSI) and if c) the PBSI indicates the particular short range transmission is a proximity beacon transmission associated with the proximity beacon service.

67. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

Let me know if you have any specific questions and I'll do my best to answer them.

Share Improve this answer Follow

answered Sep 22, 2020 at 17:30

Xlythe
2,033 1 10 14

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

The `ConnectionLifecycleCallback` parameter is the callback that will be invoked when discoverers request to connect to the advertiser. See Manage Connections for details about defining this callback.

The following example shows how to discover:

```
private void startDiscovery() {
  DiscoveryOptions discoveryOptions =
      new DiscoveryOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startDiscovery(SERVICE_ID, endpointDiscoveryCallback, discoveryOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're discovering!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We're unable to start discovering.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

68. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

69. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

> **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

70. Each of the Accused Products is capable of performing, and does perform, the operation of "then: determining if an entity or object associated with the proximity beacon service is in proximity to the mobile wireless device, by utilizing the identifier related information and the unique identifier." For example, each of the Accused Products includes and supports Google's

Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by then determining if an entity or object associated with the proximity beacon service is in proximity to the mobile wireless device, by utilizing the identifier related information and the unique identifier.

71. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share.
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code**.
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

Tips:

- On selected devices with Android 13 and up, you can also share contents from your clipboard with Quick Share. After you copy something, you get a pop-up confirmation at the bottom of the screen. Tap Devices and choose the device you want to share the content with.
- When you scan the QR code on the sending device, the receiving device appears on the sender device's screen. It automatically connects and receives the content.
- To use Quick Share, make sure Wi-Fi and Bluetooth access is granted when prompted.

https://support.google.com/android/answer/9286773?hl=en#zippy=%2Cshare-content-with-someone%2Cget-content-from-someone%2Cfix-problems-sharing-content

Not a contact, no problem.

Quick Share lets you share files with anyone nearby, even if they aren't in your contacts. Simply set the receiver's visibility to everyone or generate a QR code for your selected files, and the recipient can scan it with their device to instantly receive the content. No need to exchange contact information or adjust any settings – just scan and share.

https://www.android.com/articles/quick-share-on-android/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code || `endpointId` || SHA256(`serviceId`)[:3] || separator || `ncname`))

A. Discovery and Connection Request

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

72. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

73. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

> **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

74. Defendant has had knowledge of the ’918 Patent and of its infringement of the ’918 Patent at least through receipt of this Complaint. Alternatively, on information and belief, Defendant has had knowledge of the ’918 Patent and of its infringement of the ’918 Patent at least through receipt or knowledge of SCT’s Original Complaint in *Secure Communication*

Technologies, LLC v. Samsung Electronics Co., Ltd. et al., No. 2:24-cv-00484-JRG (the "Samsung Case"), in the U.S. District Court for the Eastern District of Texas and/or through receipt or knowledge of SCT's Infringement Contentions in the Samsung Case, in which SCT identified how the Accused Products in that case (including products that utilize Google's Android operating system) infringe claims of the '918 Patent. Alternatively, Defendant has had knowledge of the `918 Patent, and its infringement of the `918 Patent, at least since sometime before July 2, 2025, when Defendant filed IPR2025-01181 challenging the validity of the `918 Patent and referencing the Samsung Case.

75. On information and belief, despite Defendant's knowledge of the '918 Patent and of its infringement of the '918 Patent, Defendant has not sought to remedy its infringement or sought to identify any good faith belief as to why it does not infringe the '918 Patent.

76. On information and belief, Defendant's actions represented a specific intent to induce infringement of at least claims 1 and 9 of the '918 Patent. For example, Defendant offered its customers extensive customer support and instructions that instructed and encouraged its customers to infringe the '918 Patent via at least their use of the Accused Products. *See, e.g.,* materials cited in paragraphs 53 through 73 above; https://support.google.com/android/answer/9286773?hl=en and https://support.google.com/android/answer/15728591?hl=en (exemplar support pages for Quick Share); https://support.google.com/android/answer/15728092?hl=en#:~:text=Tap%20Location%20.,nearby%20devices%20on%20or%20off. (exemplar support page for Fast Pair); https://support.google.com/android/answer/15992026?hl=en, https://support.google.com/android/answer/3265955?hl=en, and

https://support.google.com/android/answer/6160491?hl=en-GB#:~:text=If%20you%20lose%20an%20Android,Is%20visible%20on%20Google%20Play (exemplar Find Hub support pages).

77. In addition, by receiving notice of the '918 Patent and Defendant's infringement thereof, Defendant obtained a subjective belief that there is a high probability that the Accused Products infringe the '918 Patent. Despite being put on notice of infringement, on information and belief Defendant has not taken actions to avoid the conduct alleged to infringe, has not offered any reasons as to why Defendant does not infringe the '918 Patent, and has not sought to remedy its infringements by offering to take a license. Defendant's failure to act reflects deliberate actions to avoid learning that the Accused Products infringe the '918 Patent and, more generally, a policy of not earnestly reviewing and respecting the intellectual property of others.

78. Since issuance of the '918 Patent, and prior to Defendant having notice of infringement, neither SCT nor its predecessors or licensees have made, offered for sale, sold, or imported a product that practices any claim of the '918 Patent or that would otherwise require marking under 35 U.S.C. § 287.

79. SCT may recover pre-suit damages for Defendant's infringement of the '918 Patent under 35 U.S.C. § 287.

80. As a result of Defendant's infringement of the '918 Patent, SCT has suffered and is owed monetary damages adequate to compensate it for the infringement under 35 U.S.C. § 284, but in no event less than a reasonable royalty.

81. Defendant's activities have been willful, egregious, wanton, and deliberate in disregard to SCT's rights, justifying a finding of willful infringement, enhanced damages under

35 U.S.C. § 284 and attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

COUNT II – INFRINGEMENT OF THE '344 PATENT

82. SCT incorporates herein the allegations made in paragraphs 1 through 81.

83. Defendant has directly infringed one or more claims of the '344 Patent, including, for example, claims 1, 29, and 30, in violation of 35 U.S.C. § 271(a) by making, using, selling, offering for sale, and/or importing into the United States infringing products including, but not limited to, the Accused Products.

84. Additionally, Defendant has indirectly infringed the '344 Patent in violation of 35 U.S.C. § 271(b) at least by inducing customers to purchase the Accused Products and/or by instructing customers how to use the Accused Products in a way that directly infringes at least claims 1, 29, and 30 of the '344 Patent.

85. Each of the Accused Products performs "[a] method comprising" the steps set forth in the paragraphs below.

86. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, at a wireless device, via a short range wireless protocol, a first plurality of beacon transmissions, each beacon transmission comprising three fields (a) a MAC address, (b) a unique identifier, wherein the unique identifier of each respective beacon transmission does not directly identify a location of a beacon transmitter, and (c) a beacon service identifier, wherein the beacon service identifier of each respective beacon transmission identifies a wireless beacon service as opposed to identifying a communication network or a device accessible using the short range wireless protocol." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by

receiving, at a wireless device, via a short range wireless protocol, a first plurality of beacon transmissions, each beacon transmission comprising three fields (a) a MAC address, (b) a unique identifier, wherein the unique identifier of each respective beacon transmission does not directly identify a location of a beacon transmitter, and (c) a beacon service identifier, wherein the beacon service identifier of each respective beacon transmission identifies a wireless beacon service as opposed to identifying a communication network or a device accessible using the short range wireless protocol.

87. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

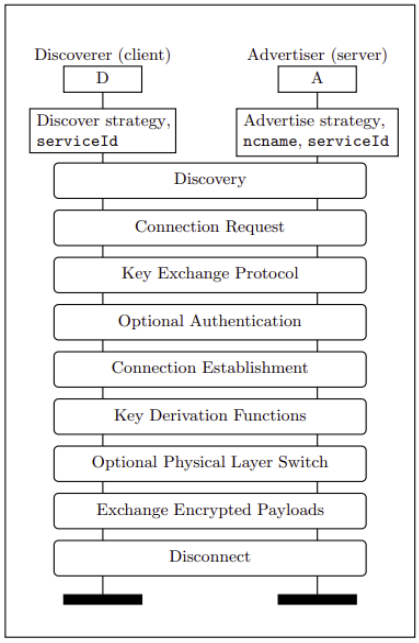


Fig. 1: The Nearby Connections API has two types of actors: the Discoverer (client) and the Advertiser (server). It uses application layer encryption and optional user authentication.

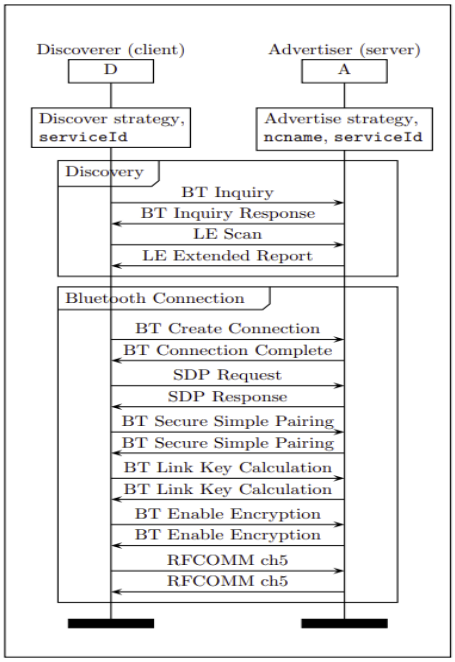


Fig. 3: Nearby Connections Request. BT is Bluetooth BR/EDR. Secure Simple Pairing provides link-layer encryption.

D Antononioli, N.O. Tippenhauer and K. Rasmussen, "Nearby threats: reversing, analyzing and attacking Google's Nearby Connections on Android," Network and Distribution Systems Security (NDSS) Symposium, 2019, San Diego CA.

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

Public Methods

public DiscoveryOptions **build** ()

public DiscoveryOptions.Builder **setLowPower** (boolean lowPower)

Sets whether to use low power. If true, only low power mediums (like BLE) will be used for discovery. By default, this option is false.

public DiscoveryOptions.Builder **setStrategy** (Strategy strategy)

Sets the discovery strategy. Must match the strategy used in `AdvertisingOptions`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/DiscoveryOptions.Builder

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

A. Discovery and Connection Request

A nearby connection is always requested using Bluetooth, regardless of the nearby connection parameters. Discovering and advertising are done in a deterministic and predictable way without using encryption. An attacker who posses a clone of the library can pretend to be any advertiser and discoverer of any application, and he can use any Bluetooth compatible device to request a connection. The Bluetooth connection uses Secure Simple Pairing (SSP) with a link key that is not authenticated and not persistent. Indeed, an attacker could insert himself as a man in the middle in the Bluetooth link and he can force the re-establishment of the link key at any time.

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

https://francozappa.github.io/publication/rearby/paper.pdf

As in the connection request phase, Wi-Fi and Bluetooth LE are not used to establish a nearby connection. We discovered that devices with the same `ncname` are allowed to connect, this means that in a nearby connection only the `endpointId` uniquely identifies a node. While testing the connection establishment phase we discovered interesting things about the `serviceId`. Any advertiser leaks its `serviceId` if queried with a generic SDP request, e.g., using `sdptool browse adv_btaddress`. Moreover, two devices from different applications can use the same `serviceId` to establish a connection. This means that it is possible to predict the `serviceId` of any application . We also discovered that the library is still using an undocumented `serviceId` named `__LEGACY_SERVICE_ID__`.

https://francozappa.github.io/publication/rearby/paper.pdf

EndpointDiscoveryCallback

On this page

Public Constructor Summary
Public Method Summary
Inherited Method Summary
Public Constructors
Public Methods

public abstract class **EndpointDiscoveryCallback** extends Object

Listener invoked during endpoint discovery.

Public Constructor Summary

EndpointDiscoveryCallback()	

Public Method Summary

abstract void	onEndpointFound(String endpointId, DiscoveredEndpointInfo info) Called when a remote endpoint is discovered.
abstract void	onEndpointLost(String endpointId) Called when a remote endpoint is no longer discoverable; only called for endpoints that previously had been passed to `onEndpointFound(String, DiscoveredEndpointInfo)`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/EndpointDiscoveryCallback

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

A. *Discovery and Connection Request*

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

Advertise and discover

Once the user has granted all required permissions, your app can begin to advertise and discover in order to find nearby devices.

First, choose a `Strategy` for your use case. The `Strategy` you select determines the connection topology for your app (one advertiser to N discoverers, or M advertisers to N discoverers).

On devices that will advertise, call `startAdvertising()` with the desired `Strategy` and a `serviceId` parameter that identifies your app.

On devices that will discover nearby advertisers, call `startDiscovery()` with the same `Strategy` and `serviceId`.

The `serviceId` value must uniquely identify your app. As a best practice, use the package name of your app (for example, `com.google.example.myapp`).

The following example shows how to advertise:

```
private void startAdvertising() {
  AdvertisingOptions advertisingOptions =
      new AdvertisingOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startAdvertising(
          getLocalUserName(), SERVICE_ID, connectionLifecycleCallback, advertisingOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're advertising!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We were unable to start advertising.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

88. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Configuration

Roles

The profile defines two roles: *Fast Pair Seeker*, and *Fast Pair Provider*. The Seeker is normally a phone, looking for a device to pair with. The Provider is a device that is advertising its presence and readiness to pair (e.g. a discoverable pair of headphones).

The Fast Pair Seeker shall use the GAP Central role. The Fast Pair Provider shall use the GAP Peripheral role.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Fast Pair uses Bluetooth Low Energy (BLE) technology to communicate with your phone. BLE is a wireless protocol that consumes very little power and has a long range. It also has a high level of security, as it encrypts the data that is transmitted between devices. Fast Pair uses BLE to send a unique identifier from your device to your phone, which then contacts Google's servers to get more information about the device, such as its name and icon. This way, you can easily recognize your device on your phone screen and pair it with a single tap.

https://communityin.oppo.com/thread/1301071443236225027

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/introduction

Model Registration

All Provider models must be registered with Google before they will work with Fast Pair. After registration, Google will distribute a Model ID and Anti-Spoofing Public/Private Key Pair. The information provided during registration is used in the pairing suggestion presented to the user, and may be used in other UX.

Example pairing suggestion (subject to change)

https://developers.google.com/nearby/fast-pair/specifications/service/modelregistration

Provider Advertising signal

Advertising: When discoverable

When the Provider device is BR/EDR discoverable (that is, in pairing mode), it shall advertise Fast Pair Model ID Data over BLE, and the BLE address shall not be rotated.

Advertising interval: When discoverable

The interval between advertisements should be no larger than 100ms (10Hz). A fast rate allows the Seeker to quickly find the Provider, even when scanning in low-power mode.

Advertising payload: Fast Pair Model ID Data

The advertisement shall contain the Service Data data type, ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0-2	`uint24`	24-bit model ID	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Advertising: When not discoverable

When not discoverable (that is, not in pairing mode), the Provider device shall advertise Fast Pair Account Data, using the following guidelines.

Advertising the account data allows Seekers nearby to recognize when a provider belongs to their account and initiate pairing without having to force the provider back into pairing mode first, which is a common cause for user complaint. Seekers will provide the opportunity for users to be able to ignore this broadcast in the case where they do not wait to pair with the provider or the broadcast is not relevant (for example, if they have already paired). Seekers will also filter out obviously bad broadcasts automatically, such as when the account data is misconfigured.

Advertising interval: When not discoverable

The interval between advertisements should be at most 250ms (4Hz).

Advertising payload: Fast Pair Account Data

The advertisement shall contain the Service Data data type, Ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0	`uint8`	Version and flags 0bVVVVFFFF • V = version • F = flags	`0x00` (reserved for future use)
1 - *varies*		Account Key Data	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

BLE address information

To prevent tracking, BLE advertising shall use the random resolvable private address (*RPA*). The address shall be rotated at minimum every 15 minutes while the device is actively advertising, and every time the state changes from not advertising to advertising. A randomized offset should be used to alter the address randomization interval.

https://developers.google.com/nearby/fast-pair/specifications/configuration

The Account Key Data contains:

Octet	Data type	Description	Value
0	`uint8`	Field length and type 0bLLLLTTTT • L = length of account key filter in bytes • T = type	0bLLLL0000 • length = 0bLLLL = *varies* • type = 0b0000 (show UI indication) or 0b0010 (hide UI indication), Account Key Filter
1 - *s*		Account Key Filter	*varies*
s + 1	`uint8`	Field length and type 0bLLLLTTTT • L = length in bytes • T = type	0b00100001 • length = 0b0010 = 2 • type = 0b0001, Salt
s + 2 - *s* + 3	`uint16`	Salt	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Salt field

The salt is a random value that is appended to account keys when building the bloom filter. This salt should be regenerated every time the RPA is updated for the Provider to avoid tracking across address rotation.

To generate the Account Key Filter using the salt:

1. Generate a random 2-byte *S*. Note that there is no "endianness" to this value, since there is no more or less significant byte — don't alter the byte order.
2. Use the 2-byte *S* as the Salt.
3. In the advertised Fast Pair Account Data, include the generated filter in the Account Key Filter field, and *S* in the Salt field.

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Keys: Account Key List

The Provider shall allocate space to store a persisted list of 128-bit Account Keys. Each Account Key allows the Provider to be recognized as belonging to a certain user account.

The list must be able to store at least five keys (that is, there must be at least 80 bytes of space dedicated to this list). Providers can optionally store more than this, they just must make sure that the keys will fit inside of their advertising packet. The exact number that can be stored will depend on how many free bytes are available in the advertising packet; see the Account Key Filter section for more information on determining how many bytes each key will take up. For example, to advertise 10 account keys, 15 bytes need to be available in the packet.But for personal devices (e.g. headphones), the number of account keys should not be greater than 5. This is to avoid the number of account keys becoming too large, and hence it could be unique and trackable.

This list is initially empty, and must be cleared if the Provider is factory-reset (if the user clears its paired device list). The list is populated as described in the Account Key characteristic section.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Account Key Filter

★ **Note:** Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

The Account Key Filter is a variable-length Bloom filter constructed as follows:

1. Let *n* be the number of account keys (*n* >= 1) in the persisted Account Key list.
2. Let *s*, the size of the filter in bytes, be (1.2**n* + 3) truncated. For example, if 1 key is persisted, *s* = 4 bytes.
   `uint8_t s = (((uint8_t)(( float )1.2 * n)) + 3);`
3. Initialize the filter *F* as an array of *s* bytes, each set to 0.
   `uint8_t F[s] = {0};`
4. For each account key *K* in the persisted Account Key list:
   a. Let *V* be concat(*K*, Salt).

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

```
// In the sample code, the size of salt is 2 bytes.
#define SALT_SIZE 2

uint8_t V[FASTPAIR_ACCOUNT_KEY_SIZE + SALT_SIZE];
for (uint8_t keyIndex = 0; keyIndex < n; keyIndex++)
  {
     // concat (K, Salt)
      fastpair_get_account_key_by_index(keyIndex, V);

      uint8_t randomSalt = (uint8_t)rand();
      V[FASTPAIR_ACCOUNT_KEY_SIZE] = randomSalt;
      ... }
```

b. Hash *V* using SHA256, obtaining a 32-byte value $H = \{H_0, ..., H_{31}\}$.

```
uint8_t H[32] = {0};
SHA256_hash_function(V, H);
```

c. Divide *H* into eight 4-byte unsigned integers in big-endian, $X = \{X_0, ..., X_7\}$, where $X_0 = 0xH_0H_1H_2H_3$.

```
     uint32_t X[8];
     for (index = 0; index < 8; index++)
     {
        X[index] = (((uint32_t)(H[index * 4])) << 24) |
                    (((uint32_t)(H[index * 4 + 1])) << 16) |
                    (((uint32_t)(H[index * 4 + 2])) << 8) |
                    (((uint32_t)(H[index * 4 + 3])) << 0);
     }
```

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

Characteristics

Fast Pair Service

The Fast Pair Provider shall have the following GATT service.

Service	UUID
Fast Pair Service	`0xFE2C`

This service shall have the following characteristics.

Fast Pair Service characteristic	Encrypted	Permissions	UUID
Model ID	No	Read	`FE2C1233-8366-4814-8EB0-01DE32100BEA`
Key-based Pairing	No	Write and notify	`FE2C1234-8366-4814-8EB0-01DE32100BEA`
Passkey	No	Write and notify	`FE2C1235-8366-4814-8EB0-01DE32100BEA`
Account Key	No	Write	`FE2C1236-8366-4814-8EB0-01DE32100BEA`

https://developers.google.com/nearby/fast-pair/specifications/characteristics

Besides the seamless connections, Fast Pair also stores details about the Bluetooth accessory on to your Google ID, so every device running that Google account can automatically identify the accessory. After being added to your Google account, the accessory also appears among other devices in Google's Find My Device app, which can be used to track the accessory's whereabouts or details of the last Android device connected in case you lose it.

https://www.slashgear.com/1454224/google-fast-pair-service-android-how-to/. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

1.1 Definition

- **Initial pairing** is the sequence of events that occur when a user pairs a device to a Google Account signed-in on phone for the very first time. In this sequence, a phone detects the advertisement from the device and displays a notification prompting the user to connect to and save the device. (In this guideline, 'device' means the Bluetooth headset or speaker instead of a reference phone).
- **Subsequent pairing** is the sequence of events that occur when a user signs into their Google Account on a new phone and attempts to pair a device already saved to their Google Account. In this sequence, the new phone recognizes that the advertised Model ID is already saved to the user's Google Account and provides a notification to expedite pairing the device to this phone.

https://developers.google.com/nearby/fast-pair/fast-pair-certification-guideline

89. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page
GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

90. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, at the wireless device via a second wireless protocol, stored information from one or more servers, the stored information relating to a particular entity or object associated with a first unique identifier, and wherein the second wireless protocol is different from the short

range wireless protocol." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by receiving, at the wireless device via a second wireless protocol, stored information from one or more servers, the stored information relating to a particular entity or object associated with a first unique identifier, and wherein the second wireless protocol is different from the short range wireless protocol.

91. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

A new way to share files

Sharing pictures, files and text between devices should be simple, fast and convenient. In 2020, we introduced Nearby Share to make it easy to share files across devices. Samsung also has its own capability called Quick Share, which is loved by their users.

Collaborating with Samsung, we're bringing the best of our sharing solutions together into a singular cross-Android solution under the Quick Share [1] name. We've integrated the experiences and created the best default, built-in option for peer-to-peer content sharing across all types of devices in the Android and Chromebook ecosystems. And to make sharing even more seamless between devices, we're working with leading PC manufacturers like LG to expand Quick Share to Windows PCs as a pre-installed app.

This means with a simple tap of the new Quick Share icon, you can see a list of available devices close by. You remain in control of your privacy, and can choose in your phone's settings who can discover your device and send files, whether it's everyone, only your contacts or just your own devices. Quick Share will start rolling out to current Nearby Share enabled devices next month.

https://blog.google/products/android/ces-2024-android-updates/

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

With Quick Share, you can immediately send and receive files from devices close to your location.

Important:

- Quick Share is available on Android 6+ devices and Chromebooks, and on selected Windows devices through an App.
- For a Galaxy device with Android 10 & One UI 2.1 or later, the settings and features can be different. Learn about Quick Share feature on Galaxy devices .
- This feature was previously known as Nearby Share. If you still find Nearby Share on some devices, Quick Share will still work.

https://support.google.com/android/answer/9286773

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share.
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code**.
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

https://support.google.com/android/answer/9286773

Adjust Quick Share settings

1. At the top of your screen, swipe down.
2. Tap Quick Share .
   - If there's no Quick Share :
     a. Tap Edit .
     b. Under "Quick Settings," add Quick Share .
3. Choose who can share with you:
   - **Your devices:** If your screen is off, your device is still visible to your other devices with the same Google Account.
   - **Contacts:** While your screen is on and unlocked, your device is visible to your nearby contacts.
   - **Everyone:** While your screen is on and unlocked, your device is visible to anyone nearby.
     - **Only for 10 minutes:** To protect your privacy, your visibility automatically switches back to your previous selection after 10 minutes.
4. To manage additional settings, like how you appear to others, at the lower left, tap **Settings**.

https://support.google.com/android/answer/9286773

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

Back up & sync device & SIM contacts

To back up and sync your device and SIM contacts, save them as Google contacts:

1. On your Android phone or tablet, open the "Settings" app.
2. Tap **Google** › **Settings for Google apps** › **Google Contacts sync** › **Also sync device and SIM contacts** › **Sync device and SIM contacts automatically.**
3. Turn on **Sync device and SIM contacts automatically.**
4. Pick the account you'd like your contacts to be saved in. Your contacts can only be automatically saved to one Google Account.

https://support.google.com/contacts/answer/9423168?hl=en

92. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account -- you can rename devices to whatever you want. Whenever the service is able to detect an accessory near other known host devices, it'll automatically prompt the user to pair with the accessory.

GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account.

GFPS can also display your accessory's battery levels in a persistent notification, aid in passing off audio sourcing (if your audio gear supports multipoint) to a different device as needed, provide an interface on your phone to control certain features on your earbuds like ANC, and helps you locate your accessory by either ringing an alarm through them or showing their last known location while paired.

https://www.pocket-lint.com/what-is-google-fast-pair-and-how-does-it-work/

Characteristic: Key-based Pairing

★ **Note:** Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

This characteristic controls the Key-based Pairing procedure. In this procedure, a certain level of trust is established by verifying that the Seeker and Provider are both in possession of a pre-shared key. The key is different in each case:

- Case 1: The pre-shared key is based on the anti-spoofing public/private key pair, and the Seeker's own public/private key pair which will change for each pairing attempt.
  - The Provider is in pairing mode.
  - The Seeker verifies that the Provider is in possession of the anti-spoofing private key.

  Note that when in pairing mode, the Provider may also of course pair in the usual way, for example, to pair with a device that does not support Fast Pair's Key-based Pairing.
- Case 2: The pre-shared key is one of the account keys.
  - The Provider is usually not in pairing mode. (But this is not a requirement—The Provider should support using an account key even when in pairing mode.)
  - The Seeker and Provider each verify that the other is in possession of the account key.

Since both cases are extremely similar, except for which pre-shared key is used, they are combined in procedure.

https://developers.google.com/nearby/fast-pair/specifications/characteristics#KeyBasedPairing

93. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

94. Each of the Accused Products is capable of performing, and does perform, the operation of "causing selection of one or more of the unique identifiers from the first plurality of beacon transmissions, by filtering the beacon transmissions which include a particular beacon service identifier indicating that a particular received beacon transmission is associated with a particular wireless beacon service." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by causing selection of one or more of the unique identifiers from the first plurality of beacon transmissions, by filtering the beacon transmissions which include a particular beacon service identifier indicating that a particular received beacon transmission is associated with a particular wireless beacon service.

95. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

Let me know if you have any specific questions and I'll do my best to answer them.

Share Improve this answer Follow

answered Sep 22, 2020 at 17:30

Xlythe
2,033 ● 1 ● 10 ● 14

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

The `ConnectionLifecycleCallback` parameter is the callback that will be invoked when discoverers request to connect to the advertiser. See Manage Connections for details about defining this callback.

The following example shows how to discover:

```
private void startDiscovery() {
  DiscoveryOptions discoveryOptions =
      new DiscoveryOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startDiscovery(SERVICE_ID, endpointDiscoveryCallback, discoveryOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're discovering!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We're unable to start discovering.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

96. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

97. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

> ★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

> ★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

98. Each of the Accused Products is capable of performing, and does perform, the operation of "taking first further action relating to the stored information, if the first unique identifier is present among the selected one or more unique identifiers." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by taking first further action relating to the stored information, if the first unique identifier is present among the selected one or more unique identifiers.

99. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share.
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code**.
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

Tips:

- On selected devices with Android 13 and up, you can also share contents from your clipboard with Quick Share. After you copy something, you get a pop-up confirmation at the bottom of the screen. Tap Devices and choose the device you want to share the content with.
- When you scan the QR code on the sending device, the receiving device appears on the sender device's screen. It automatically connects and receives the content.
- To use Quick Share, make sure Wi-Fi and Bluetooth access is granted when prompted.

https://support.google.com/android/answer/9286773?hl=en#zippy=%2Cshare-content-with-someone%2Cget-content-from-someone%2Cfix-problems-sharing-content

Not a contact, no problem.

Quick Share lets you share files with anyone nearby, even if they aren't in your contacts. Simply set the receiver's visibility to everyone or generate a QR code for your selected files, and the recipient can scan it with their device to instantly receive the content. No need to exchange contact information or adjust any settings – just scan and share.

https://www.android.com/articles/quick-share-on-android/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

A. Discovery and Connection Request

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

100. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

101. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

> ★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

> ★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

102. Defendant has had knowledge of the ’344 Patent and of its infringement of the ’344 Patent at least through receipt of this Complaint. Alternatively, on information and belief, Defendant has had knowledge of the ’344 Patent and of its infringement of the ’344 Patent at least through receipt or knowledge of SCT’s Original Complaint in the Samsung Case and/or through receipt or knowledge of SCT’s Infringement Contentions in the Samsung Case, in which SCT identified how the Accused Products in that case (including products that utilize Google’s Android operating system) infringe claims of the ’344 Patent. Alternatively, Defendant has had knowledge of the ’344 Patent, and its infringement of the ’344 Patent, at least since sometime before July 2, 2025, when Defendant filed IPR2025-01182 challenging the validity of the ’344 Patent and referencing the Samsung Case.

103. On information and belief, despite Defendant’s knowledge of the ’344 Patent and of its infringement of the ’344 Patent, Defendant has not sought to remedy its infringement or sought to identify any good faith belief as to why it does not infringe the ’344 Patent.

104. On information and belief, Defendant’s actions represented a specific intent to induce infringement of at least claims 1, 29, and 30 of the ’344 Patent. For example, Defendant offered its customers extensive customer support and instructions that instructed and encouraged its customers to infringe the ’344 Patent via at least their use of the Accused Products. *See, e.g.,* materials cited in paragraphs 85 through 101 above; https://support.google.com/android/answer/9286773?hl=en and https://support.google.com/android/answer/15728591?hl=en (exemplar support pages for Quick Share);

https://support.google.com/android/answer/15728092?hl=en#:~:text=Tap%20Location%20.,near

by%20devices%20on%20or%20off. (exemplar support page for Fast Pair); https://support.google.com/android/answer/15992026?hl=en, https://support.google.com/android/answer/3265955?hl=en, and https://support.google.com/android/answer/6160491?hl=en-GB#:~:text=If%20you%20lose%20an%20Android,Is%20visible%20on%20Google%20Play (exemplar Find Hub support pages).

105. In addition, by receiving notice of the '344 Patent and Defendant's infringement thereof, Defendant obtained a subjective belief that there is a high probability that the Accused Products infringe the '344 Patent. Despite being put on notice of infringement, on information and belief Defendant has not taken actions to avoid the conduct alleged to infringe, has not offered any reasons as to why Defendant does not infringe the '344 Patent, and has not sought to remedy its infringements by offering to take a license. Defendant's failure to act reflects deliberate actions to avoid learning that the Accused Products infringe the '344 Patent and, more generally, a policy of not earnestly reviewing and respecting the intellectual property of others.

106. Since issuance of the '344 Patent, and prior to Defendant having notice of infringement, neither SCT nor its predecessors or licensees have made, offered for sale, sold, or imported a product that practices any claim of the '344 Patent or that would otherwise require marking under 35 U.S.C. § 287.

107. SCT may recover pre-suit damages for Defendant's infringement of the '344 Patent under 35 U.S.C. § 287.

108. As a result of Defendant's infringement of the '344 Patent, SCT has suffered and is owed monetary damages adequate to compensate it for the infringement under 35 U.S.C. § 284, but in no event less than a reasonable royalty.

109. Defendant's activities have been willful, egregious, wanton, and deliberate in disregard to SCT's rights, justifying a finding of willful infringement, enhanced damages under 35 U.S.C. § 284 and attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

COUNT III – INFRINGEMENT OF THE '971 PATENT

110. SCT incorporates herein the allegations made in paragraphs 1 through 109.

111. Defendant has directly infringed one or more claims of the '971 Patent, including, for example, claims 1, 37, and 50, in violation of 35 U.S.C. § 271(a) by making, using, selling, offering for sale, and/or importing into the United States infringing products including, but not limited to, the Accused Products.

112. Additionally, Defendant has indirectly infringed the '971 Patent in violation of 35 U.S.C. § 271(b) at least by inducing customers to purchase the Accused Products and/or by instructing customers how to use the Accused Products in a way that directly infringes at least claims 1, 37, and 50 of the '971 Patent.

113. Each of the Accused Products performs "[a] method comprising" the steps set forth in the paragraphs below.

114. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, at a wireless device, via a short range wireless protocol, a first plurality of beacon transmissions, each beacon transmission comprising three fields (a) a MAC address, (b) a unique identifier, wherein the unique identifier of each respective beacon transmission does not directly identify a location of a beacon transmitter, and (c) a beacon service identifier, wherein the beacon service identifier of each respective beacon transmission identifies that the unique identifier is relevant to stored information associated with a wireless beacon service, wherein the stored information is not otherwise accessible to the wireless device utilizing the short range

wireless protocol." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by receiving, at a wireless device, via a short range wireless protocol, a first plurality of beacon transmissions, each beacon transmission comprising three fields (a) a MAC address, (b) a unique identifier, wherein the unique identifier of each respective beacon transmission does not directly identify a location of a beacon transmitter, and (c) a beacon service identifier, wherein the beacon service identifier of each respective beacon transmission identifies that the unique identifier is relevant to stored information associated with a wireless beacon service, wherein the stored information is not otherwise accessible to the wireless device utilizing the short range wireless protocol.

115. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

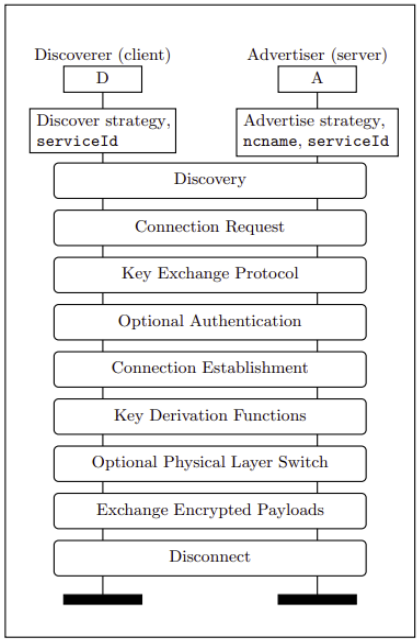


Fig. 1: The Nearby Connections API has two types of actors: the Discoverer (client) and the Advertiser (server). It uses application layer encryption and optional user authentication.

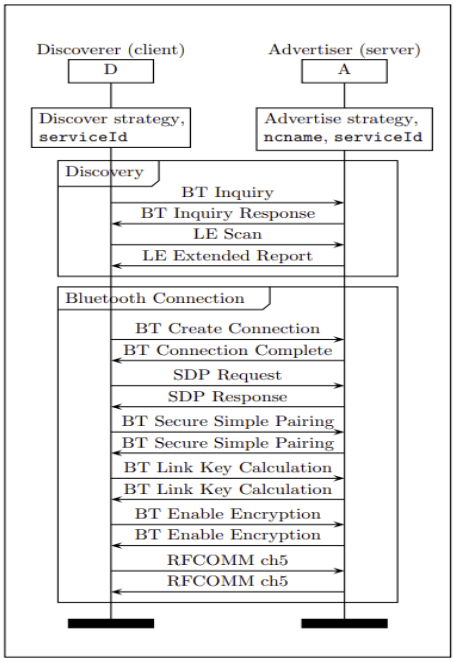


Fig. 3: Nearby Connections Request. BT is Bluetooth BR/EDR. Secure Simple Pairing provides link-layer encryption.

D Antononioli, N.O. Tippenhauer and K. Rasmussen, "Nearby threats: reversing, analyzing and attacking Google's Nearby Connections on Android," Network and Distribution Systems Security (NDSS) Symposium, 2019, San Diego CA.

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

Public Methods

public DiscoveryOptions **build** ()

public DiscoveryOptions.Builder **setLowPower** (boolean lowPower)

Sets whether to use low power. If true, only low power mediums (like BLE) will be used for discovery. By default, this option is false.

public DiscoveryOptions.Builder **setStrategy** (Strategy strategy)

Sets the discovery strategy. Must match the strategy used in `AdvertisingOptions`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/DiscoveryOptions.Builder

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

A. Discovery and Connection Request

A nearby connection is always requested using Bluetooth, regardless of the nearby connection parameters. Discovering and advertising are done in a deterministic and predictable way without using encryption. An attacker who posses a clone of the library can pretend to be any advertiser and discoverer of any application, and he can use any Bluetooth compatible device to request a connection. The Bluetooth connection uses Secure Simple Pairing (SSP) with a link key that is not authenticated and not persistent. Indeed, an attacker could insert himself as a man in the middle in the Bluetooth link and he can force the re-establishment of the link key at any time.

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

https://francozappa.github.io/publication/rearby/paper.pdf

As in the connection request phase, Wi-Fi and Bluetooth LE are not used to establish a nearby connection. We discovered that devices with the same `ncname` are allowed to connect, this means that in a nearby connection only the `endpointId` uniquely identifies a node. While testing the connection establishment phase we discovered interesting things about the `serviceId`. Any advertiser leaks its `serviceId` if queried with a generic SDP request, e.g., using `sdptool browse adv_btaddress`. Moreover, two devices from different applications can use the same `serviceId` to establish a connection. This means that it is possible to predict the `serviceId` of any application . We also discovered that the library is still using an undocumented `serviceId` named `__LEGACY_SERVICE_ID__`.

https://francozappa.github.io/publication/rearby/paper.pdf

EndpointDiscoveryCallback

On this page

Public Constructor Summary
Public Method Summary
Inherited Method Summary
Public Constructors
Public Methods

public abstract class **EndpointDiscoveryCallback** extends Object

Listener invoked during endpoint discovery.

Public Constructor Summary

EndpointDiscoveryCallback()	

Public Method Summary

abstract void	onEndpointFound(String endpointId, DiscoveredEndpointInfo info) Called when a remote endpoint is discovered.
abstract void	onEndpointLost(String endpointId) Called when a remote endpoint is no longer discoverable; only called for endpoints that previously had been passed to `onEndpointFound(String, DiscoveredEndpointInfo)`.

https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/EndpointDiscoveryCallback

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code || `endpointId` || SHA256(`serviceId`)[:3] || separator || `ncname`))

A. Discovery and Connection Request

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

Advertise and discover

Once the user has granted all required permissions, your app can begin to advertise and discover in order to find nearby devices.

First, choose a `Strategy` for your use case. The `Strategy` you select determines the connection topology for your app (one advertiser to N discoverers, or M advertisers to N discoverers).

On devices that will advertise, call `startAdvertising()` with the desired `Strategy` and a `serviceId` parameter that identifies your app.

On devices that will discover nearby advertisers, call `startDiscovery()` with the same `Strategy` and `serviceId`.

The `serviceId` value must uniquely identify your app. As a best practice, use the package name of your app (for example, `com.google.example.myapp`).

The following example shows how to advertise:

```
private void startAdvertising() {
  AdvertisingOptions advertisingOptions =
      new AdvertisingOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startAdvertising(
          getLocalUserName(), SERVICE_ID, connectionLifecycleCallback, advertisingOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're advertising!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We were unable to start advertising.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

116. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Configuration

Roles

The profile defines two roles: *Fast Pair Seeker*, and *Fast Pair Provider*. The Seeker is normally a phone, looking for a device to pair with. The Provider is a device that is advertising its presence and readiness to pair (e.g. a discoverable pair of headphones).

The Fast Pair Seeker shall use the GAP Central role. The Fast Pair Provider shall use the GAP Peripheral role.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Fast Pair uses Bluetooth Low Energy (BLE) technology to communicate with your phone. BLE is a wireless protocol that consumes very little power and has a long range. It also has a high level of security, as it encrypts the data that is transmitted between devices. Fast Pair uses BLE to send a unique identifier from your device to your phone, which then contacts Google's servers to get more information about the device, such as its name and icon. This way, you can easily recognize your device on your phone screen and pair it with a single tap.

https://communityin.oppo.com/thread/1301071443236225027

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

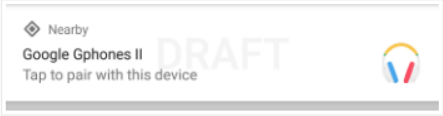
GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/introduction

Model Registration

All Provider models must be registered with Google before they will work with Fast Pair. After registration, Google will distribute a Model ID and Anti-Spoofing Public/Private Key Pair. The information provided during registration is used in the pairing suggestion presented to the user, and may be used in other UX.



Example pairing suggestion (subject to change)

https://developers.google.com/nearby/fast-pair/specifications/service/modelregistration

Provider Advertising signal

Advertising: When discoverable

When the Provider device is BR/EDR discoverable (that is, in pairing mode), it shall advertise Fast Pair Model ID Data over BLE, and the BLE address shall not be rotated.

Advertising interval: When discoverable

The interval between advertisements should be no larger than 100ms (10Hz). A fast rate allows the Seeker to quickly find the Provider, even when scanning in low-power mode.

Advertising payload: Fast Pair Model ID Data

The advertisement shall contain the Service Data data type, ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0-2	`uint24`	24-bit model ID	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Advertising: When not discoverable

When not discoverable (that is, not in pairing mode), the Provider device shall advertise Fast Pair Account Data, using the following guidelines.

Advertising the account data allows Seekers nearby to recognize when a provider belongs to their account and initiate pairing without having to force the provider back into pairing mode first, which is a common cause for user complaint. Seekers will provide the opportunity for users to be able to ignore this broadcast in the case where they do not wait to pair with the provider or the broadcast is not relevant (for example, if they have already paired). Seekers will also filter out obviously bad broadcasts automatically, such as when the account data is misconfigured.

Advertising interval: When not discoverable

The interval between advertisements should be at most 250ms (4Hz).

Advertising payload: Fast Pair Account Data

The advertisement shall contain the Service Data data type, Ibid., § 1.11. The UUID shall be the Fast Pair Service UUID of `0xFE2C`. The service data shall contain the following:

Octet	Data type	Description	Value
0	`uint8`	Version and flags 0bVVVVFFFF • V = version • F = flags	`0x00` (reserved for future use)
1 - *varies*		Account Key Data	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider. *See also*

https://developers.google.com/nearby/fast-pair/specifications/introduction, and

https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

BLE address information

To prevent tracking, BLE advertising shall use the random resolvable private address (*RPA*). The address shall be rotated at minimum every 15 minutes while the device is actively advertising, and every time the state changes from not advertising to advertising. A randomized offset should be used to alter the address randomization interval.

https://developers.google.com/nearby/fast-pair/specifications/configuration

The Account Key Data contains:

Octet	Data type	Description	Value
0	`uint8`	Field length and type 0bLLLLTTTT • L = length of account key filter in bytes • T = type	0bLLLL0000 • length = 0bLLLL = *varies* • type = 0b0000 (show UI indication) or 0b0010 (hide UI indication), Account Key Filter
1 - *s*		Account Key Filter	*varies*
s + 1	`uint8`	Field length and type 0bLLLLTTTT • L = length in bytes • T = type	0b00100001 • length = 0b0010 = 2 • type = 0b0001, Salt
s + 2 - *s* + 3	`uint16`	Salt	*varies*

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Salt field

The salt is a random value that is appended to account keys when building the bloom filter. This salt should be regenerated every time the RPA is updated for the Provider to avoid tracking across address rotation.

To generate the Account Key Filter using the salt:

1. Generate a random 2-byte *S*. Note that there is no "endianness" to this value, since there is no more or less significant byte — don't alter the byte order.
2. Use the 2-byte *S* as the Salt.
3. In the advertised Fast Pair Account Data, include the generated filter in the Account Key Filter field, and *S* in the Salt field.

https://developers.google.com/nearby/fast-pair/specifications/service/provider

Keys: Account Key List

The Provider shall allocate space to store a persisted list of 128-bit Account Keys. Each Account Key allows the Provider to be recognized as belonging to a certain user account.

The list must be able to store at least five keys (that is, there must be at least 80 bytes of space dedicated to this list). Providers can optionally store more than this, they just must make sure that the keys will fit inside of their advertising packet. The exact number that can be stored will depend on how many free bytes are available in the advertising packet; see the Account Key Filter section for more information on determining how many bytes each key will take up. For example, to advertise 10 account keys, 15 bytes need to be available in the packet.But for personal devices (e.g. headphones), the number of account keys should not be greater than 5. This is to avoid the number of account keys becoming too large, and hence it could be unique and trackable.

This list is initially empty, and must be cleared if the Provider is factory-reset (if the user clears its paired device list). The list is populated as described in the Account Key characteristic section.

https://developers.google.com/nearby/fast-pair/specifications/configuration

Account Key Filter

★ **Note:** Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

The Account Key Filter is a variable-length Bloom filter constructed as follows:

1. Let *n* be the number of account keys (*n* >= 1) in the persisted Account Key list.
2. Let *s*, the size of the filter in bytes, be (1.2**n* + 3) truncated. For example, if 1 key is persisted, *s* = 4 bytes.
   `uint8_t s = (((uint8_t)(( float )1.2 * n)) + 3);`
3. Initialize the filter *F* as an array of *s* bytes, each set to 0.
   `uint8_t F[s] = {0};`
4. For each account key *K* in the persisted Account Key list:
   a. Let *V* be concat(*K*, Salt).

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

```
// In the sample code, the size of salt is 2 bytes.
#define SALT_SIZE 2

uint8_t V[FASTPAIR_ACCOUNT_KEY_SIZE + SALT_SIZE];
for (uint8_t keyIndex = 0; keyIndex < n; keyIndex++)
  {
     // concat (K, Salt)
      fastpair_get_account_key_by_index(keyIndex, V);

      uint8_t randomSalt = (uint8_t)rand();
      V[FASTPAIR_ACCOUNT_KEY_SIZE] = randomSalt;
      ... }
```

b. Hash *V* using SHA256, obtaining a 32-byte value $H = \{H_0, ..., H_{31}\}$.

```
uint8_t H[32] = {0};
SHA256_hash_function(V, H);
```

c. Divide *H* into eight 4-byte unsigned integers in big-endian, $X = \{X_0, ..., X_7\}$, where $X_0 = 0xH_0H_1H_2H_3$.

```
     uint32_t X[8];
     for (index = 0; index < 8; index++)
     {
        X[index] = (((uint32_t)(H[index * 4])) << 24) |
                    (((uint32_t)(H[index * 4 + 1])) << 16) |
                    (((uint32_t)(H[index * 4 + 2])) << 8) |
                    (((uint32_t)(H[index * 4 + 3])) << 0);
     }
```

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

Characteristics

Fast Pair Service

The Fast Pair Provider shall have the following GATT service.

Service	UUID
Fast Pair Service	`0xFE2C`

This service shall have the following characteristics.

Fast Pair Service characteristic	Encrypted	Permissions	UUID
Model ID	No	Read	`FE2C1233-8366-4814-8EB0-01DE32100BEA`
Key-based Pairing	No	Write and notify	`FE2C1234-8366-4814-8EB0-01DE32100BEA`
Passkey	No	Write and notify	`FE2C1235-8366-4814-8EB0-01DE32100BEA`
Account Key	No	Write	`FE2C1236-8366-4814-8EB0-01DE32100BEA`

https://developers.google.com/nearby/fast-pair/specifications/characteristics

Besides the seamless connections, Fast Pair also stores details about the Bluetooth accessory on to your Google ID, so every device running that Google account can automatically identify the accessory. After being added to your Google account, the accessory also appears among other devices in Google's Find My Device app, which can be used to track the accessory's whereabouts or details of the last Android device connected in case you lose it.

https://www.slashgear.com/1454224/google-fast-pair-service-android-how-to/. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

1.1 Definition

- **Initial pairing** is the sequence of events that occur when a user pairs a device to a Google Account signed-in on phone for the very first time. In this sequence, a phone detects the advertisement from the device and displays a notification prompting the user to connect to and save the device. (In this guideline, 'device' means the Bluetooth headset or speaker instead of a reference phone).
- **Subsequent pairing** is the sequence of events that occur when a user signs into their Google Account on a new phone and attempts to pair a device already saved to their Google Account. In this sequence, the new phone recognizes that the advertised Model ID is already saved to the user's Google Account and provides a notification to expedite pairing the device to this phone.

https://developers.google.com/nearby/fast-pair/fast-pair-certification-guideline

117. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page
GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

> ★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

118. Each of the Accused Products is capable of performing, and does perform, the operation of "receiving, at the wireless device via a second wireless protocol, the stored information from one or more servers, the stored information relating to a particular entity or object associated with a first unique identifier, and wherein the second wireless protocol is different from

the short range wireless protocol." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by receiving, at the wireless device via a second wireless protocol, the stored information from one or more servers, the stored information relating to a particular entity or object associated with a first unique identifier, and wherein the second wireless protocol is different from the short range wireless protocol.

119. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

A new way to share files

Sharing pictures, files and text between devices should be simple, fast and convenient. In 2020, we introduced Nearby Share to make it easy to share files across devices. Samsung also has its own capability called Quick Share, which is loved by their users.

Collaborating with Samsung, we're bringing the best of our sharing solutions together into a singular cross-Android solution under the Quick Share [1] name. We've integrated the experiences and created the best default, built-in option for peer-to-peer content sharing across all types of devices in the Android and Chromebook ecosystems. And to make sharing even more seamless between devices, we're working with leading PC manufacturers like LG to expand Quick Share to Windows PCs as a pre-installed app.

This means with a simple tap of the new Quick Share icon, you can see a list of available devices close by. You remain in control of your privacy, and can choose in your phone's settings who can discover your device and send files, whether it's everyone, only your contacts or just your own devices. Quick Share will start rolling out to current Nearby Share enabled devices next month.

https://blog.google/products/android/ces-2024-android-updates/

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

With Quick Share, you can immediately send and receive files from devices close to your location.

Important:

- Quick Share is available on Android 6+ devices and Chromebooks, and on selected Windows devices through an App.
- For a Galaxy device with Android 10 & One UI 2.1 or later, the settings and features can be different. Learn about Quick Share feature on Galaxy devices .
- This feature was previously known as Nearby Share. If you still find Nearby Share on some devices, Quick Share will still work.

https://support.google.com/android/answer/9286773

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share .
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code** .
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

https://support.google.com/android/answer/9286773

Adjust Quick Share settings

1. At the top of your screen, swipe down.
2. Tap Quick Share.
   - If there's no Quick Share:
     a. Tap Edit.
     b. Under "Quick Settings," add Quick Share.
3. Choose who can share with you:
   - **Your devices:** If your screen is off, your device is still visible to your other devices with the same Google Account.
   - **Contacts:** While your screen is on and unlocked, your device is visible to your nearby contacts.
   - **Everyone:** While your screen is on and unlocked, your device is visible to anyone nearby.
     - **Only for 10 minutes:** To protect your privacy, your visibility automatically switches back to your previous selection after 10 minutes.
4. To manage additional settings, like how you appear to others, at the lower left, tap **Settings**.

https://support.google.com/android/answer/9286773

Safely share and receive files with those around you

Nearby Share was built with privacy at its core, so you can share and receive files with peace of mind. Now you don't have to worry about exchanging contact information, because Nearby Share allows you to both send and receive files anonymously. It also allows you to adjust your privacy settings from your phone's Quick Settings at any time. You can be "hidden," visible to "some contacts" or visible to "all contacts," so you never receive files that you didn't ask for.

https://blog.google/products/android/nearby-share/

Back up & sync device & SIM contacts

To back up and sync your device and SIM contacts, save them as Google contacts:

1. On your Android phone or tablet, open the "Settings" app.
2. Tap **Google** › **Settings for Google apps** › **Google Contacts sync** › **Also sync device and SIM contacts** › **Sync device and SIM contacts automatically.**
3. Turn on **Sync device and SIM contacts automatically.**
4. Pick the account you'd like your contacts to be saved in. Your contacts can only be automatically saved to one Google Account.

https://support.google.com/contacts/answer/9423168?hl=en

120. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account -- you can rename devices to whatever you want. Whenever the service is able to detect an accessory near other known host devices, it'll automatically prompt the user to pair with the accessory.

> ***GFPS logs down a history of the accessories you've connected to recognized phones, tablets, and Chromebooks to your Google account.***

GFPS can also display your accessory's battery levels in a persistent notification, aid in passing off audio sourcing (if your audio gear supports multipoint) to a different device as needed, provide an interface on your phone to control certain features on your earbuds like ANC, and helps you locate your accessory by either ringing an alarm through them or showing their last known location while paired.

https://www.pocket-lint.com/what-is-google-fast-pair-and-how-does-it-work/

Characteristic: Key-based Pairing

★ **Note:** Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

This characteristic controls the Key-based Pairing procedure. In this procedure, a certain level of trust is established by verifying that the Seeker and Provider are both in possession of a pre-shared key. The key is different in each case:

- Case 1: The pre-shared key is based on the anti-spoofing public/private key pair, and the Seeker's own public/private key pair which will change for each pairing attempt.
  - The Provider is in pairing mode.
  - The Seeker verifies that the Provider is in possession of the anti-spoofing private key.

  Note that when in pairing mode, the Provider may also of course pair in the usual way, for example, to pair with a device that does not support Fast Pair's Key-based Pairing.
- Case 2: The pre-shared key is one of the account keys.
  - The Provider is usually not in pairing mode. (But this is not a requirement—The Provider should support using an account key even when in pairing mode.)
  - The Seeker and Provider each verify that the other is in possession of the account key.

Since both cases are extremely similar, except for which pre-shared key is used, they are combined in procedure.

https://developers.google.com/nearby/fast-pair/specifications/characteristics#KeyBasedPairing

121. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

122. Each of the Accused Products is capable of performing, and does perform, the operation of "causing selection of one or more of the unique identifiers from the first plurality of beacon transmissions, by filtering the beacon transmissions which include a particular beacon service identifier indicating that a particular received beacon transmission is associated with a particular wireless beacon service." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by causing selection of one or more of the unique identifiers from the first plurality of beacon transmissions, by filtering the beacon transmissions which include a particular beacon service identifier indicating that a particular received beacon transmission is associated with a particular wireless beacon service.

123. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

I'm the lead of Nearby Share and Nearby Connections at Google.

Yup, Nearby Share is built on top of the Nearby Connections API. Share is the user-facing component of the two, and adds a concept of 'Identity' that Nearby Connections lacks. The reliability & throughput improvements are all a part of Nearby Connections, so users of the API should benefit.

Let me know if you have any specific questions and I'll do my best to answer them.

Share Improve this answer Follow

answered Sep 22, 2020 at 17:30

Xlythe
2,033 ● 1 ● 10 ● 14

https://stackoverflow.com/questions/63991333/how-is-googles-nearby-share-implemented

The `ConnectionLifecycleCallback` parameter is the callback that will be invoked when discoverers request to connect to the advertiser. See Manage Connections for details about defining this callback.

The following example shows how to discover:

```
private void startDiscovery() {
  DiscoveryOptions discoveryOptions =
      new DiscoveryOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startDiscovery(SERVICE_ID, endpointDiscoveryCallback, discoveryOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're discovering!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We're unable to start discovering.
          });
}
```

https://developers.google.com/nearby/connections/android/discover-devices

124. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also* https://developers.google.com/nearby/fast-pair/specifications/introduction, and https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

125. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
    - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
    - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

> ★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

★ **Note:** It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

★ **Note:** Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

126. Each of the Accused Products is capable of performing, and does perform, the operation of "taking first further action relating to the stored information, if the first unique identifier is present among the selected one or more unique identifiers." For example, each of the Accused Products includes and supports Google's Nearby Share/Quick Share, Fast Pair, and Find

Hub features. As shown in the following paragraphs, each of these accused features in the Accused Products performs this operation by taking first further action relating to the stored information, if the first unique identifier is present among the selected one or more unique identifiers.

127. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Nearby Share/Quick Share feature and associated functionality.

Share & receive content on your Android device

You can share and get contents with someone when you use Quick Share.

Share content with someone

1. Open the content, like a photo or webpage from any app.
   - Quick Share has no limit on content size or number of files, but the app that you share from might.
2. Tap Share > Quick Share.
3. Make sure the device you want to share with is nearby.
4. Tap the device you want to share with.
   - If the device you want to share with isn't listed, you can also share your file with a QR code. To generate a QR code, tap **Use QR code**.
   - Nearby Android devices can scan the QR code to get connected and receive the files.
5. Wait for the other device to accept your sharing request.
   - To cancel an in-progress transfer, tap the same device again.

Tips:

- On selected devices with Android 13 and up, you can also share contents from your clipboard with Quick Share. After you copy something, you get a pop-up confirmation at the bottom of the screen. Tap Devices and choose the device you want to share the content with.
- When you scan the QR code on the sending device, the receiving device appears on the sender device's screen. It automatically connects and receives the content.
- To use Quick Share, make sure Wi-Fi and Bluetooth access is granted when prompted.

https://support.google.com/android/answer/9286773?hl=en#zippy=%2Cshare-content-with-someone%2Cget-content-from-someone%2Cfix-problems-sharing-content

Not a contact, no problem.

Quick Share lets you share files with anyone nearby, even if they aren't in your contacts. Simply set the receiver's visibility to everyone or generate a QR code for your selected files, and the recipient can scan it with their device to instantly receive the content. No need to exchange contact information or adjust any settings – just scan and share.

https://www.android.com/articles/quick-share-on-android/

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

So I was messing around with the nearby share feature that google launched last year and I noticed that if you connect your Gmail and have a profile picture , when you try the nearby share feature the other person can see your profile picture. How is this possible if you haven't made a connection between you and the other person yet? Does google save the MAC address of the user and then pulls it off a server or something? I'm pretty confused. I found this post (How is Google's nearby share implemented?) that is broadly talking about the implementation but only thing mentioned is that they tried to bring more of an identity to this feature. So how does the contact info show without connecting? Is it that when someone is on your contact list google has just saved their device name or MAC address and that's how it knows what profile pic and Gmail to use/show on the nearby devices?

Each device generates a certificate (containing the device's name, photo, etc) and uploads them to a Google server. The device then uploads a list of contacts who are allowed to download that certificate.

Devices periodically download certificates they have access to. Other than the start/end timestamp, the data in the certificate is encrypted. You need to see the decryption key over Bluetooth in order to parse the certificate, after which you can now attach a name to the Bluetooth advertisement you just saw.

Share Improve this answer Follow

answered Nov 17, 2021 at 18:36

Xlythe
2,033 ● 1 ● 10 ● 14

So the certificate doesn't include a MAC address, cause i would think it needed that to identify the exact device? Basically the key is sent via Bluetooth without a visible connection(interaction or prompt before the main connection) by the other users and then pulls the data from a server to show the photo, name, etc? – Chris_Gr Nov 18, 2021 at 20:28

The BLE advertisement encodes the decryption key as part of the service data. There is also a MAC address in the encoded certificate, but it's optional since you can connect to the device via BLE L2CAP or GATT. – Xlythe Nov 19, 2021 at 20:39

https://stackoverflow.com/questions/69991721/how-does-nearby-share-know-some-user-information-before-making-a-connection

An example of a nearby connection request is shown in Figure 3. The advertiser (server) advertises a strategy, a `serviceId` and a `ncname`. The discoverer (client) discovers a strategy and a `serviceId`. To find each other, both have to look for the same `serviceId`, and use the same strategy. The server to be discovered changes its Bluetooth name (btname) and sets custom LE extended report. The btname is changed to a string that depends on the strategy, the `endpointId`, the `serviceId`, and the `ncname`. btname is computed as follows:

btname = unpad(b64encode(strategy_code ‖ `endpointId` ‖ SHA256(`serviceId`)[:3] ‖ separator ‖ `ncname`))

A. Discovery and Connection Request

We manage all the Bluetooth operations with the `bluetooth` Python module. The discovery phase begins using the `discover_bt` function. This function returns the Bluetooth's name (btname) of all the discoverable devices that are in range. The custom discoverer detects the presence of any advertiser by looking at btnames. It extracts the strategy, `endpointId` and `ncname` using in reverse the btname formula of Section III-A. The custom advertiser starts an RFCOMM server on port 5 using `BluetoothSocket(RFCOMM)`. Then it computes the custom `uuid` based on the `serviceId` and it starts an SDP server advertising the `uuid` using `serviceId` as the name of the SDP service. The custom advertiser waits for the discoverers and manages each of them with separate sockets. All the Bluetooth connections are encrypted at the link layer using the shared secret computed from the secure simple pairing (SSP).

https://francozappa.github.io/publication/rearby/paper.pdf

128. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Fast Pair feature and associated functionality.

Google Fast Pair Service

Introduction

The Google Fast Pair Service (*GFPS*) utilizes Bluetooth Low Energy (*BLE*) to discover nearby Bluetooth devices without using significant phone battery, enabling "magical" scenarios based on device proximity.

Features

GFPS is aimed at facilitating the pairing of Bluetooth and BLE devices, such as speakers, headphones, car kits, mice and keyboards, with as little user interaction required as possible. By implementing the following spec, Google will continue to release additional features that build upon it. This includes:

1. Displaying a half page notification when the device is in pairing mode to facilitate easy initial pairing. Additionally companion apps are easily marketed to users.
2. Associating the device with the user's account after the initial pairing has completed.
3. Displaying a subsequent pairing notification when the device is turned on and near another phone, tablet, or desktop that the user owns, so that the user does not need to know how to put the device back into pairing mode before pairing with their other devices.
4. Associating a personalized name with the device.
5. Battery notifications are displayed for the headphones.
6. Shows device details in Android 11+.
7. Ability for users to locate a lost headset or buds.
8. Offline pairing is available for low-network situations.
9. Support Audio switch to seamlessly transition headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).
10. Support Hearable Controls to provide better access controls for important Hearable features.

https://developers.google.com/nearby/fast-pair/specifications/introduction

Audio Switch

As users increasingly utilize multiple audio source devices to perform their daily tasks, there's growing demand for a simpler solution to manage headset use across devices. Audio switch seamlessly transitions headset connections between devices based on user activity (e.g. starting a movie) and prioritized events (e.g. an incoming call).

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass

Advertising payload

The Provider shall include its current connection status in the advertisement, built on top of the Fast Pair Account data described in Advertising: When not discoverable.

Note that the version of table 4.2 is 0x1.

https://developers.google.com/nearby/fast-pair/specifications/extensions/sass. *See also*

https://developers.google.com/nearby/fast-pair/specifications/introduction, and

https://www.slashgear.com/android-fast-pair-new-features-google-pixel-buds-2-27618482/.

Account Key Filter

Note: Google recommends implementing the Cryptographic Test Cases to ease verification of these requirements.

The advertised Account Key Filter allows a Seeker to quickly check whether a Provider might possess a certain account key (with a low false-positive probability, on average much less than 0.5%), before further interactions. The Seeker may automatically connect and attempt to start the procedure when it sees a filter being broadcast with type 0, i.e. showing UI indication, that potentially contains one of its account keys, so as to reduce the rate of false positives further. In some situations, the Provider may want to be recognized by the Seeker while not ready for pairing. One example is that when buds get put back into case, we want to stop showing the subsequent pairing notification since that pairing could be rejected by the headset.

https://developers.google.com/nearby/fast-pair/specifications/service/provider#AccountKeyFilter

129. As shown herein, the Accused Products are capable of performing, and do perform, this operation via their Find Hub feature and associated functionality.

Find Hub Network Accessory Specification

On this page

GATT Specification
Authentication
Operations
Advertised frames
Ephemeral identifier (EID) computation
...

v1.3

The Find Hub Network (FHN) accessory specification defines an end-to-end encrypted approach for tracking beaconing Bluetooth Low Energy (BLE) devices. This page describes FHN as an extension to the Fast Pair specification. Providers should enable this extension if they have devices that are compatible with FHN and are willing to enable location tracking for those devices.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Google's Find My Device becomes Find Hub amid expansion

Sarah Perez
13 May 2025 • 1 min read

Android users will have more ways to find their devices and other items, Google announced on Tuesday during the Android Show, a week before Google I/O 2025. The company says its Find My Device feature, which allows Android users to locate lost phones and other devices, will become known as "Find Hub" as it rolls out support for more partners, satellite-based finding capabilities, and airline partnerships.

https://au.finance.yahoo.com/news/googles-device-becomes-hub-amid-171936846.html

Unique Features of Google Find My Device

1. Global Positioning: Leveraging a Vast Network of Devices

Unlike other Bluetooth tracker apps, Google's Find My Device capitalizes on its extensive network of over a billion Android devices worldwide. This vast ecosystem allows the service to utilize these devices as "signal nodes," aiding users in locating lost items.

- **How It Works:**
  - When your tracker (such as those compatible with Google's network) disconnects from your phone, it enters a "lost mode."
  - Nearby Android devices detect the tracker's Bluetooth signal and securely upload the location information to Google's servers.
  - You can then view the item's last known location in real-time via the Find My Device app.

This global positioning capability ensures that even if your item is far from your phone, other Android devices can assist in locating it, offering coverage that surpasses traditional Bluetooth trackers.

https://www.seinxon.com/blogs/blog-posts/google-find-my-device-vs-other-bluetooth-tracker-apps

2.3.1 Advertising PDUs

The following advertising physical channel PDU Types are called advertising PDUs:

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

These PDUs are sent by the Link Layer in the Advertising state and received by a Link Layer in the Scanning state or Initiating state. The ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND, and ADV_SCAN_IND PDUs are called "legacy advertising PDUs". The ADV_EXT_IND, AUX_ADV_IND, AUX_SYNC_IND, and AUX_CHAIN_IND PDUs are called "extended advertising PDUs". Advertising events using legacy advertising PDUs are called "legacy advertising events".

2.3.1.1 ADV_IND

The Payload field of the ADV_IND PDU is shown in Figure 2.6. The PDU shall be used in connectable and scannable undirected advertising events. The TxAdd in the advertising physical channel PDU header indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The ChSel field in the advertising physical channel PDU header shall be set to 1 if the advertiser supports the LE Channel Selection Algorithm #2 feature (see Section 4.5.8.3).

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 2.6: ADV_IND PDU payload

The Payload consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field, if not empty, shall contain Advertising Data from the advertiser's Host.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

4.4.2 Advertising state

The Link Layer shall enter the Advertising state when directed by the Host. When placed in the Advertising state, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events, periodic advertising events, or both.

https://www.bluetooth.com/specifications/specs/core-specification-5-3/

How does crowdsourcing work?

Android devices participating in the Find Hub network use Bluetooth to scan for nearby items. If they detect your items, they securely send the location where they detected the items to Find Hub. Your Android device does the same to help others find their lost items when it detects them nearby.

End-to-end encryption

The Find Hub network encrypts the locations of your items using a unique key that only you can access by entering your Android device's PIN, pattern, or password.

This end-to-end encryption, which is backed by the same technology used by Google Password Manager to secure your passwords, ensures that the locations of your items are private from Google. They're only visible to you and those you share your items with in Find Hub.

Important: If you haven't set a PIN, pattern, or password on your Android device, you must set one to take advantage of Find Hub network. Until then, by default, the network uses your Android device to help others find their items and your Android device stores encrypted recent locations for itself and connected accessories with Google to help you find your devices. To take advantage of the Find Hub network and have the best offline finding experience, set a PIN, pattern, or password on your Android device.

https://support.google.com/product-documentation/answer/14796936?hl=en

- **Data Safeguards:** We've implemented protections that help ensure the privacy of everyone participating in the network and the crowdsourced location data that powers it.
  - **Location data is end-to-end encrypted.** When Android devices participating in the network report the location of a Bluetooth tag, the location is end-to-end encrypted using a key that is only accessible to the Bluetooth tag owner and anyone the owner has shared the tag with in the Find Hub app. Only the Bluetooth tag owner (and those they've chosen to share access with) can decrypt and view the tag's location. With end-to-end encrypted location data, Google cannot decrypt, see, or otherwise use the location data.
  - **Private, crowdsourced location reports.** These end-to-end encrypted locations are contributed to the Find Hub network in a manner that does not allow Google to identify the owners of the nearby Android devices that provided the location data. And when the Find Hub network shows the location and timestamp to the Bluetooth tag's owner to help them find their belongings, no other information about the nearby Android devices that contributed the data is included.

https://security.googleblog.com/2024/04/find-my-device-network-security-privacy-protections.html

Data processed by the network

In addition to end-to-end encrypted locations, the Find Hub network processes data such as temporary device identifiers, timestamps when your device detects an item and when you request the location of your lost items, and info about the Fast Pair accessories that you have paired to your device or share with others. The Find Hub network uses this data for reasons like implementing features, delivering location info to the right person when an item is lost, and providing privacy and anti-abuse protections, such as the aggregation feature described below. Importantly, Google can't identify you when your Android device shares the location of a detected item.

Individuals using the Find Hub network to find their lost items don't receive any information from the network other than the location where their item was detected and approximately when their item was last seen.

https://support.google.com/product-documentation/answer/14796936

Current ephemeral identifier (device information code 0x0B)

The Provider can use the *current ephemeral identifier (code 0x0B)* to report the current EID and clock value when the Provider is provisioned for FHN, to sync the Seeker in case of a clock drift (for example, due to drained battery). Otherwise, the Seeker initiates a more expensive and less reliable connection for this purpose.

Octet	Data Type	Description	Value
0	uint8	Device information event	0x03
1	uint8	Current ephemeral identifier	0x0B
2 - 3	uint16	Additional data length	0x0018 or 0x0024
4 - 7	byte array	Clock value	Example: 0x13F9EA80
8 - 19 or 31	byte array	Current EID	Example: 0x112233445566778899001122334455667788990 0

Table 13: *Device information event: clock sync.*

Encryption with EID

★ **Note:** The following details for encryption and decryption with EIDs are only included for completeness of the specification. Only the Seeker implements these, not the Provider.

To encrypt a message `m`, a sighter (having read `Rx` from the beacon) would do the following:

1. Choose a random number `s` in `Fp`, as defined in the EID computation section.
2. Compute `S = s * G`.
3. Compute `R = (Rx, Ry)` by substitution in the curve equation and picking an arbitrary `Ry` value out of the possible results.
4. Compute the 256-bit AES key `k = HKDF-SHA256((s * R)x)` where `(s * R)x` is the `x` coordinate of the curve multiplication result. Salt isn't specified.
5. Let `URx` and `LRx` be the upper and lower 80-bits of `Rx`, respectively, in big-endian format. In a similar way, define `USx` and `LSx` for `S`.
6. Compute `nonce = LRx || LSx`.
7. Compute `(m', tag) = AES-EAX-256-ENC(k, nonce, m)`.
8. Send `(URx, Sx, m', tag)` to the owner, possibly through an untrusted remote service.

Decryption of values encrypted with EID

The owner's client, which is in possession of the EIK and the rotation period exponent, decrypts the message as follows:

1. Given `URx`, obtain the beacon time counter value on which `URx` is based. This can be done by the owner's client computing `Rx` values for beacon time counter values for the recent past and near future.
2. Given the beacon time counter value on which `URx` is based, compute the anticipated value of `r` as defined in the EID computation section.
3. Compute `R = r * G`, and verify a match to the value of `URx` provided by the sighter.
4. Compute `S = (Sx, Sy)` by substitution in the curve equation and picking an arbitrary `Sy` value out of the possible results.
5. Compute `k = HKDF-SHA256((r * S)x)` where `(r * S)x` is the `x` coordinate of the curve multiplication result.
6. Compute `nonce = LRx || LSx`.
7. Compute `m = AES-EAX-256-DEC(k, nonce, m', tag)`.

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

Advertised frames

After provisioning, the Provider is expected to advertise FHN frames at least once every 2 seconds. If Fast Pair frames are advertised, the Provider should interleave the FHN frames within the regular Fast Pair advertisements. For example, every two seconds, the Provider should advertise seven Fast Pair advertisements and one FHN advertisement.

The conducted Bluetooth transmit power for FHN advertisements should be set to at least 0 dBm.

Note: It is recommended to advertise the FHN frames even when the device is in low power mode, as this allows finding the device if lost. For some devices, it might not be possible to advertise when the device is turned off due to the battery life implications. In such cases, it is strongly recommended to periodically wake up the device and advertise for a short period of time (e.g. wake up every 10 minutes and advertise for 10 seconds).

The FHN frame carries a public key used to encrypt location reports by any supporting client that contributes to the crowdsourcing network. Two types of elliptic curve keys are available: a 160-bit key that fits legacy BLE 4 frames, or 256-bit key that requires BLE 5 with extended advertising capabilities. The Provider's implementation determines which curve is used.

Note: Using 256-bit keys means that older phones that don't support BLE 5 can't scan and report for advertising devices, thus reducing the size of the network.

An FHN frame is structured as follows.

Octet	Value	Description
0	0x02	Length
1	0x01	Flags data type value
2	0x06	Flags data
3	0x18 or 0x19	Length
4	0x16	Service data data type value
5	0xAA	16-bit service UUID
6	0xFE	...
7	0x40 or 0x41	FHN frame type with unwanted tracking protection mode indication
8..27		20-byte ephemeral identifier
28		Hashed flags

https://developers.google.com/nearby/fast-pair/specifications/extensions/fmdn

130. Defendant has had knowledge of the '971 Patent and of its infringement of the '971 Patent at least through receipt of this Complaint. Alternatively, on information and belief, Defendant has had knowledge of the '971 Patent and of its infringement of the '971 Patent at least

through receipt or knowledge of SCT's Original Complaint in the Samsung Case and/or through receipt or knowledge of SCT's Infringement Contentions in the Samsung Case, in which SCT identified how the Accused Products in that case (including products that utilize Google's Android operating system) infringe claims of the '971 Patent. Alternatively, Defendant has had knowledge of the ``971 Patent, and its infringement of the ``971 Patent, at least since sometime before July 2, 2025, when Defendant filed IPR2025-01183 challenging the validity of the ``971 Patent and referencing the Samsung Case.

131. On information and belief, despite Defendant's knowledge of the '971 Patent and of its infringement of the '971 Patent, Defendant has not sought to remedy its infringement or sought to identify any good faith belief as to why it does not infringe the '971 Patent.

132. On information and belief, Defendant's actions represented a specific intent to induce infringement of at least claims 1, 37, and 50 of the '971 Patent. For example, Defendant offered its customers extensive customer support and instructions that instructed and encouraged its customers to infringe the '971 Patent via at least their use of the Accused Products. *See, e.g.,* materials cited in paragraphs 113 through 129 above; https://support.google.com/android/answer/9286773?hl=en and https://support.google.com/android/answer/15728591?hl=en (exemplar support pages for Quick Share); https://support.google.com/android/answer/15728092?hl=en#:~:text=Tap%20Location%20.,nearby%20devices%20on%20or%20off. (exemplar support page for Fast Pair); https://support.google.com/android/answer/15992026?hl=en, https://support.google.com/android/answer/3265955?hl=en, and https://support.google.com/android/answer/6160491?hl=en-

GB#:~:text=If%20you%20lose%20an%20Android,Is%20visible%20on%20Google%20Play (exemplar Find Hub support pages).

133. In addition, by receiving notice of the ’971 Patent and Defendant’s infringement thereof, Defendant obtained a subjective belief that there is a high probability that the Accused Products infringe the ’971 Patent. Despite being put on notice of infringement, on information and belief Defendant has not taken actions to avoid the conduct alleged to infringe, has not offered any reasons as to why Defendant does not infringe the ’971 Patent, and has not sought to remedy its infringements by offering to take a license. Defendant’s failure to act reflects deliberate actions to avoid learning that the Accused Products infringe the ’971 Patent and, more generally, a policy of not earnestly reviewing and respecting the intellectual property of others.

134. Since issuance of the ’971 Patent, and prior to Defendant having notice of infringement, neither SCT nor its predecessors or licensees have made, offered for sale, sold, or imported a product that practices any claim of the ’971 Patent or that would otherwise require marking under 35 U.S.C. § 287.

135. SCT may recover pre-suit damages for Defendant’s infringement of the ’971 Patent under 35 U.S.C. § 287.

136. As a result of Defendant’s infringement of the ’971 Patent, SCT has suffered and is owed monetary damages adequate to compensate it for the infringement under 35 U.S.C. § 284, but in no event less than a reasonable royalty.

137. Defendant’s activities have been willful, egregious, wanton, and deliberate in disregard to SCT’s rights, justifying a finding of willful infringement, enhanced damages under 35 U.S.C. § 284 and attorneys’ fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

DEMAND FOR A JURY TRIAL

Pursuant to Rule 38 of the Federal Rules of Civil Procedure, SCT demands a trial by jury on all issues triable of right by a jury.

PRAYER FOR RELIEF

WHEREFORE, SCT respectfully requests that this Court enter judgment in its favor and grant the following relief:

a. A judgment that Defendant has directly infringed one or more claims of each of the Patents-in-Suit;

b. A judgment and order requiring Defendant to pay SCT past and future damages under 35 U.S.C. § 284, including for supplemental damages arising from any continuing post-verdict infringement for the time between trial and entry of the final judgment with an accounting, as needed, as provided by 35 U.S.C. § 284;

c. A judgment and order that Defendant has willfully infringed the Patents-in-Suit and requiring Defendant to pay SCT enhanced damages under 35 U.S.C. § 284 and attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285;

d. A judgment and order requiring Defendant to pay SCT reasonable ongoing royalties on a going-forward basis after final judgment;

e. A judgment and order requiring Defendant to pay SCT pre-judgment and post-judgment interest on the damages award;

f. A judgment and order requiring Defendant to pay SCT's costs; and

g. Such other and further relief as the Court may deem just and proper.

SCT Exhibit 2016
IPR2025-01181

Dated: August 4, 2025

Respectfully submitted,

/s/ Timothy E. Grochocinski
Timothy E. Grochocinski
Illinois Bar No. 6295055
tim@nelbum.com
Charles Austin Ginnings
Illinois Bar No. 6343678
austin@nelbum.com
Taryn N. Trusty
Illinois Bar No. 6344409
taryn@nelbum.com
NELSON BUMGARDNER CONROY PC
745 McClintock Drive, Suite 340
Burr Ridge, Illinois 60527
708.675.1974 (telephone)

Brent Bumgardner
Texas Bar No. 00795272
brent@nelbum.com
Christopher G. Granaghan
Texas Bar No. 24078585
chris@nelbum.com
NELSON BUMGARDNER CONROY P.C.
3131 West Seventh Street, Suite 300
Fort Worth, Texas 76107
Telephone: (817) 377-9111

Attorneys for Plaintiff
Secure Communication Technologies, LLC