Alexander Heinrich*, Milan Stute*, Tim Kornhuber, and Matthias Hollick

Who Can *Find My* Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System

Abstract: Overnight, Apple has turned its hundreds-of-million-device ecosystem into the world's largest crowd-sourced location tracking network called offline finding (OF). OF leverages online finder devices to detect the presence of missing offline devices using Bluetooth and report an approximate location back to the owner via the Internet. While OF is not the first system of its kind, it is the first to commit to strong privacy goals. In particular, OF aims to ensure finder anonymity, untrackability of owner devices, and confidentiality of location reports. This paper presents the first comprehensive security and privacy analysis of OF. To this end, we recover the specifications of the closed-source OF protocols by means of reverse engineering. We experimentally show that unauthorized access to the location reports allows for accurate device tracking and retrieving a user's top locations with an error in the order of 10 meters in urban areas. While we find that OF's design achieves its privacy goals, we discover two distinct design and implementation flaws that can lead to a location correlation attack and unauthorized access to the location history of the past seven days, which could deanonymize users. Apple has partially addressed the issues following our responsible disclosure. Finally, we make our research artifacts publicly available.

Keywords: Apple, Bluetooth, location privacy, reverse engineering, trackings tags, user identification

DOI Editor to enter DOI

Received ..; revised ..; accepted ...

***Corresponding Author: Alexander Heinrich:** Secure Mobile Networking Lab, Technical University of Darmstadt, Germany, E-mail: aheinrich@seemoo.tu-darmstadt.de

***Corresponding Author: Milan Stute:** Secure Mobile Networking Lab, Technical University of Darmstadt, Germany, E-mail: mstute@seemoo.tu-darmstadt.de

Tim Kornhuber: Secure Mobile Networking Lab, Technical University of Darmstadt, Germany, E-mail: tkornhuber@seemoo.tu-darmstadt.de

Matthias Hollick: Secure Mobile Networking Lab, Technical University of Darmstadt, Germany, E-mail: mhollick@seemoo.tu-darmstadt.de

1 Introduction

In 2019, Apple introduced *offline finding (OF)*, a proprietary crowd-sourced location tracking system for offline devices. The basic idea behind OF is that so-called *finder* devices can detect the presence of other *lost* offline devices using Bluetooth Low Energy (BLE) and use their Internet connection to report an approximate location back to the *owner*. Apple's OF network consists of "hundreds of millions" of devices [4], making it the currently largest crowd-sourced location tracking system in existence. We expect the network to grow as OF will officially support the tracking of non-Apple devices in the future [6]. Regardless of its size, the system has sparked considerable interest and discussion within the broader tech and security communities [28, 29] as Apple makes strong security and privacy claims supported by new cryptographic primitives that other commercial systems are lacking [51]. In particular, Apple claims that it cannot access location reports, finder identities are not revealed, and BLE advertisements cannot be used to track devices [35]. Apple has yet to provide ample proof for their claims as, until today, only selected components have been publicized [4, 6, 35].

Contribution. This paper challenges Apple's security and privacy claims and examines the system design and implementation for vulnerabilities. To this end, we first analyze the involved OF system components on macOS and iOS using reverse engineering and present the proprietary protocols involved during *losing*, *searching*, and *finding* devices. In short, devices of one owner agree on a set of so-called rolling public–private key pairs. Devices without an Internet connection, i.e., without cellular or Wi-Fi connectivity, emit BLE advertisements that encode one of the rolling public keys. Finder devices overhearing the advertisements encrypt their current location under the rolling public key and send the location report to a central Apple-run server. When searching for a lost device, another owner device queries the central server for location reports with a set of known rolling public keys of the lost device. The owner can decrypt the reports using the corresponding private key and retrieve the location.

Based on our analysis, we assess the security and privacy of the OF system. We find that the overall design achieves Apple's specific goals. However, we discovered two distinct design and implementation vulnerabilities that seem to be outside of Apple's threat model but can have severe consequences for the users. First, the OF design allows Apple to correlate different owners' locations if their locations are reported by the same finder, effectively allowing Apple to construct a social graph. Second, malicious macOS applications can retrieve and decrypt the OF location reports of the last seven days for all its users and for *all* of their devices as cached rolling advertisement keys are stored on the file system in cleartext. We demonstrate that the latter vulnerability is exploitable and verify that the accuracy of the retrieved reports—in fact—allows the attacker to locate and identify their victim with high accuracy. We have shared our findings with Apple via responsible disclosure, who have meanwhile fixed one issue via an OS update (CVE-2020-9986, cf. *Responsible Disclosure* section for details). We summarize our key contributions.

- We provide a comprehensive specification of the OF protocol components for losing, searching, and finding devices. Our PoC implementation allows for tracking non-Apple devices via Apple's OF network.
- We experimentally evaluate the accuracy of real-world location reports for different forms of mobility (by car, train, and on foot). We show that (1) a walking user's path can be tracked with a mean error of less than 30 m in a metropolitan area and (2) the top locations of a user such as home and workplace can be inferred reliably and precisely (error in the order of 10 m) from a one-week location trace.
- We discover a design flaw in OF that lets Apple correlate the location of multiple owners if the same finder submits the reports. This would jeopardize location privacy for all other owners if only a single location became known.
- We discover that a local application on macOS can effectively circumvent Apple's restrictive location API [5] and access the user's location history without their consent, allowing for device tracking and user identification.
- We open-source our PoC implementation and experimental data (cf. *Availability* section).

Outline. The remainder of this paper is structured as follows. § 2 and § 3 provide background information about OF and the involved technology. § 4 outlines our adversary model. § 5 summarizes our reverse engineering methodology. § 6 describes the OF protocols and components in detail. § 7 evaluates the accuracy of OF location reports. § 8 assesses the security and privacy of Apple's OF design and implementation. § 9 and § 10 report two discovered vulnerabilities and propose our mitigations. § 11 reviews related work. Finally, § 12 concludes this work.

2 Background

This section gives a brief introduction to BLE and elliptic curve cryptography (ECC) as they are the basic building blocks for OF. We then cover relevant Apple platform internals.

2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) [19] is designed for small battery-powered devices such as smartwatches and fitness trackers with low data rates. Devices can broadcast BLE advertisements to inform nearby devices about their presence. The maximum BLE advertisement payload size is 31 bytes [19]. Apple heavily relies on custom BLE advertisements to announce their proprietary services such as AirDrop and bootstrap their protocols over Wi-Fi or Apple Wireless Direct Link (AWDL) [21, 36, 48]. OF devices also use BLE advertisements to inform nearby finders about their presence [6].

2.2 Elliptic Curve Cryptography

OF employs elliptic curve cryptography (ECC) for encrypting location reports. ECC is a public-key encryption scheme that uses operations on elliptic curve (EC) over finite fields. An EC is a curve over a finite field that contains a known generator (or base point) G. A private key in ECC is a random number in the finite field of the used curve. The public key is the result of the point multiplication of the generator G with the private key. The result is an X–Y coordinate on the curve. The NIST P-224 curve [39], which is used by OF [6], provides a security level of 112 bit.

2.3 Apple Platform Internals

We briefly introduce the terms keychain and iCloud as they are relevant for Apple's OF implementation.

Keychain. All Apple operating systems (OSs) use a keychain as a database to store secrets such as passwords, keys, and trusted Transport Layer Security (TLS) root certificates. The keychain is used by system services such as AirDrop [48] and third-party applications to store login information, tokens, and other

secrets. Every keychain item may contain a *keychain access group*. This group is used to identify which application can access which keychain items. Access policies are implemented via *entitlement* files embedded into signed application binaries. A system process prevents the execution of processes with unauthorized entitlements, e.g., a third-party application trying to access a system-owned keychain item. This security mechanism can be disabled on jailbroken iOS devices or by deactivating macOS system integrity protection (SIP), which helps extracting keys and secrets used by Apple's system services.

iCloud. iCloud is an umbrella term for all Apple services handling online data storage and synchronization via Apple's servers. All *owner* devices signed in to the same Apple account can synchronize themselves via iCloud. OF uses the iCloud keychain to share rolling advertisement keys across all owner devices. The synchronization is required to retrieve and decrypt the location reports from potential finders on any of the owner devices [4, 35].

3 Apple Offline Finding Overview

Apple introduced OF in 2019 for iOS 13, macOS 10.15, and watchOS 6 [10]. OF enables locating Apple devices without an Internet connection and promises to operate in a privacy-preserving manner. In 2020, Apple announced to support third-party BLE-enabled devices to be tracked by the OF network [11] and released a protocol specification for their integration [6]. We found that this public specification is incomplete concerning the overall OF system. Within this paper, we focus on our recovered specification that was partly validated by the accessory specification [6].

In the following, we give a brief overview of how OF works and introduce the different roles of devices. Fig. 1 depicts the interplay of the roles and protocols involved in OF. In particular, OF involves (1) initial pairing of owner devices, (2) broadcasting BLE advertisements that contain a rolling public key, (3) uploading encrypted location reports to Apple's servers, and (4) retrieving the location reports on owner devices. The terminology of the roles below has been derived from the official documentation [6].

Owner devices. Owner devices share a common Apple ID and can use the *Find My* application on macOS and iOS to search for any devices of the same owner.

Lost devices. Devices that determine to be in a lost state start sending out BLE advertisements with a public key to be discovered by finder devices. Apple devices

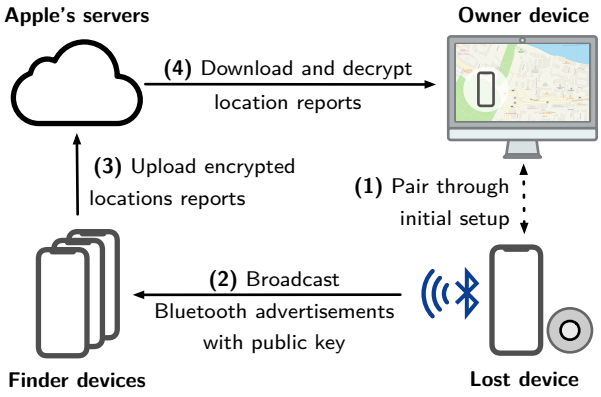

Fig. 1. Simplified offline finding (OF) workflow.

are considered to be lost when they lose Internet connectivity. Third-party accessories [6] are small battery-powered devices that can be attached to a personal item and are set up through an owner device. Accessories determine to be *lost* when they lose their BLE connection to the owner device.

Finder devices. Finder devices form the core of the OF network. As of 2020, only iPhones and iPads with a GPS module are offering finder capabilities. Finder devices can discover lost devices and accessories by scanning for BLE advertisements. Upon receiving an OF advertisement, a finder creates an end-to-end encrypted location report that includes its current location and sends it to Apple's servers.

Apple's servers. Apple's servers store OF location reports submitted by finder devices. Owner devices can fetch those reports and decrypt them locally.

4 Adversary Model

OF exposes several interfaces that might be targeted by attackers. In this section, we identify these potentially vulnerable interfaces and devise a comprehensive adversary model that will guide the rest of this paper. We first detail the four sub-models, summarized in Tab. 1, and we specify them by their assumptions, goals, and capabilities following [23]. Then, we motivate the subsequent analysis of OF protocols and components based on these models.

First of all, we consider adversaries on either of OF's communication channels (cf. (2)–(4) in Fig. 1). In particular, a proximity-based adversary has access to BLE advertisements **(A2)**, and a network-based adversary can modify traffic between OF devices and Apple's servers **(A3)**. Also, we consider a zero-permission application running with user privileges on an owner/lost de-

Table 1. Adversary models considered throughout and guiding this paper. We assume that neither adversary has direct access to cryptographic secrets or can break cryptographic primitives.

Model	Assumptions	Goals	Capabilities
Local application **(A1)**	(1) User-installed application on lost/owner devices that is either reviewed or notarized. (2) Zero-permission. (3) No privilege escalation exploits.	(1) Determine location of OF devices without asking for permission. (2) Track user by accessing current or historic location data.	(1) Communicate with any server over the Internet. (2) Read/write files that are accessible by the user and not restricted through sandboxing.
Proximity-based **(A2)**	(1) In BLE communication range of OF device. (2) Control one or more BLE transceivers to cover a larger area.	(1) Access location of lost devices or personally linkable data. (2) Track lost devices in larger areas (e.g., shopping center or airport). (3) DoS against OF service.	(1) Track devices based on advertisement content. (2) Record and replay advertisements at different locations. (3) Fabricate new advertisements.
Network-based **(A3)**	(1) MitM position between Apple and OF devices. (2) Cannot break TLS.	(1) Access location of reported lost devices. (2) Identify reported devices. (3) Identify lost devices.	(1) Redirect traffic to a different host. (2) Read, intercept, redirect, or modify traffic.
Service operator **(A4)**	(1) Apple as the service provider. (2) Controls the OF server infrastructure.	(1) Locate individuals and their lost devices. (2) Correlate locations to create a social graph.	(1) Access to all encrypted OF reports and their metadata. (2) Add, remove, or modify reports.

vice that wants to infer the user's current location. The application may be distributed inside or outside[1] of Apple's official app stores **(A1)**. Finally, we also consider Apple as the service operator as an adversary that has access to all encrypted location reports and might try to infer any information based on the report metadata such as submission times and finder identifiers **(A4)**. Note that Apple uses its iCloud keychain service for initial device pairing and key synchronization (cf. (1) in Fig. 1). Apple provides detailed information about its keychain [4], which appears to withstand professional forensics analyses [1]. Therefore, we assume that the pairing process is secure throughout this paper.

To conduct a security and privacy analysis based on these models, we need to understand OF in detail. To this end, we reverse engineer the protocols involved in loosing, finding, and searching devices (cf. (2)–(4) in Fig. 1) in § 6. Based on our understanding of OF, we conduct a security and privacy analysis of the BLE communication **(A2)**, the server communication **(A3)**, and storage of encrypted reports and cryptographic keys **(A1/A4)** in § 8.

1 On macOS, applications can be distributed outside of Apple's app store. Those applications are not reviewed [3]. However, Apple recommends submitting those application to their notarization service, which checks application for malicious code [8]. macOS displays a warning and asks for user consent before executing non-notarized applications.

5 Methodology

Our analysis of OF required a comprehensive understanding of the implemented protocols by Apple. Our methodology follows previous works analyzing the Apple ecosystem [21, 36, 44, 45, 48], while providing new insights into the reverse engineering process. We started this research with the beta releases of macOS 10.15 and iOS 13, the first Apple OSs to support OF. During that time, no official documentation from Apple was available regarding the OF design or implementation. Therefore, we used reverse engineering tools such as system log analysis, static binary analysis, and network traffic analysis. In addition, we implemented an OF prototype to validate our findings. Some of our findings, such as the BLE advertisement format and cryptographic primitives, were later confirmed by Apple's specification for third-party accessories [6].

System Logging. To get a first overview of OS internals, we used the system logging facility on macOS. It aggregates applications and kernel events, and can access the same events from a USB-attached iOS device. We can filter logs by process or keyword and adjust the log level for more verbose output. By using a special configuration profile [27], macOS will show logs that are normally redacted. On iOS, this option is only available with a jailbreak [14].

Binary analysis. We use binary analysis to understand the closed-source OF protocols. Many Apple binaries have been written in Objective-C, which uses message dispatch to resolve methods at runtime. There-

fore, Objective-C binaries include method and instance variable names as part of the dispatch table. This simplifies identifying interesting code paths and segments, e.g., those responsible for parsing BLE packets. Unfortunately, most OF code is written in the newer Swift programming language. Swift methods are statically called by their program address and, therefore, do not require an entry in the symbol table, i.e., the symbol names may be stripped by the compiler. Additionally, the Swift compiler adds several checks to achieve type safety, which clutters the compiled code and makes it hard to follow the program logic. However, dynamically linked frameworks and libraries must keep function names in the symbol table, facilitating the identification of interesting code segments. Furthermore, dynamic analysis methods aid in understanding the control flow and access function parameters at runtime. By hooking functions with a dynamic instrumentation tool such as Frida [40], we can, e.g., access cryptographic keys used by system processes as shown in [45].

Network analysis. We can identify a service's protocols by monitoring network interfaces, which helps understand the information exchange with external parties. OF uses two protocols: BLE for advertisements and HTTPS for server communication. To understand the embedded custom protocols and payloads, we rely on two sets of tools. For BLE, we use BTLEmap [31] to capture all BLE advertisements. As we already know the basic frame format of Apple's custom advertisements from related work [21, 36], we were able to identify OF as a new subtype. HTTPS proxies such as [50] decrypt HTTPS sessions by masquerading as both HTTP client and server and using self-signed TLS certificates. To access OF-related traffic, we disabled *certificate pinning*, which OF clients use for all server communication.

6 Apple Offline Finding in Detail

This section describes and discusses the technical details of Apple's OF system. In reference to Fig. 1, we (1) explain the involved cryptography and the key exchange during initial device pairing, and then explain the protocols implementing (2) *losing*, (3) *finding*, (4) *searching* for devices.

In short, devices and accessories in lost mode send out BLE advertisements containing a public key. Finder devices receive them, encrypt their location by using the public key, and upload a report to Apple's servers. This results in an end-to-end encrypted location report that cannot be read by Apple or any other third-party that does not have access to the owner's private keys. In the following, we explain the cryptography in use, the protocols involved in losing, searching, and finding devices, as well as a brief description of the system's implementation on iOS and macOS.

6.1 Cryptography

OF employs ECC [6]. In the following, we explain the key generation and derivation mechanisms and the cryptographic algorithms used for encryption and decryption.

Master Beacon and Advertisement Keys. Initially, each owner device generates a private–public key pair (d_0, p_0) on the NIST P-224 curve and a 32-byte symmetric key SK_0 that together form the *master beacon key*. Those keys are never sent out via BLE and are used to derive the rolling advertisement keys included in the BLE advertisements.

OF makes device tracking hard by regularly changing the contents of the BLE advertisements. In particular, OF uses the concept of *rolling* keys that can be deterministically derived if one knows the initial input keys (d_0, p_0) and SK_0 but are otherwise unlinkable. OF iteratively calculates the *advertisement keys* (d_i, p_i) for $i > 0$ as follows using the ANSI X.963 key derivation function (KDF) with SHA-256 [33] and a generator G of the NIST P-224 curve:

$$SK_i = \mathrm{KDF}(SK_{i-1}, \text{“update”}, 32) \tag{1}$$

$$(u_i, v_i) = \mathrm{KDF}(SK_i, \text{“diversify”}, 72) \tag{2}$$

$$d_i = (d_0 * u_i) + v_i \tag{3}$$

$$p_i = d_i * G \tag{4}$$

Equation (1) derives a new symmetric key from the last used symmetric key with 32 bytes length. Equation (2) derives the so-called “anti-tracking” keys u_i and v_i from the new symmetric key with a length of 36 bytes each. Finally, Eqs. (3) and (4) create the advertisement key pair via EC point multiplication using the anti-tracking keys and the master beacon key d_0.

Key Synchronization. All owner devices need to access the advertisement keys to download and decrypt location reports. Therefore, OF synchronizes the master beacon keys via iCloud in a property list file encrypted under Advanced Encryption Standard in Galois/Counter Mode (AES-GCM). The decryption key for the file is stored in the iCloud keychain under the label “Beacon Store.”

Encryption. The BLE advertisements sent out by a lost device contain an EC public key p_i. A finder device that receives such an advertisement determines

its current location and encrypts the location with p_i. OF employs Elliptic Curve Integrated Encryption Scheme (ECIES) that performs an ephemeral Elliptic Curve Diffie-Hellmann (ECDH) key exchange to derive a shared secret and encrypt the report [37]. In particular, the finder's encryption algorithm works as follows:

(1) Generate a new ephemeral key (d', p') on the NIST P-224 curve for a received OF advertisement.
(2) Perform ECDH using the ephemeral private key d' and the advertised public key p_i to generate a shared secret.
(3) Derive a symmetric key with ANSI X.963 KDF on the shared secret with the advertised public key as entropy and SHA-256 as the hash function.
(4) Use the first 16 bytes as the encryption key e'.
(5) Use the last 16 bytes as an initialization vector (IV).
(6) Encrypt the location report under e' and the IV with AES-GCM.

The ephemeral public key p' and the authentication tag of AES-GCM are part of the uploaded message, as shown in Fig. 2. All location reports are identified by an id, which is a SHA-256 hash of p_i.

Decryption. An owner device that retrieves encrypted location reports follows the inverse of the encryption procedure. First, the owner device selects the proper advertisement keys (d_i, p_i) based on the hashed p_i of the location report. Second, it performs the ECDH key exchange with the finder's ephemeral public key p' and the lost device's private key d_i to compute the symmetric key e' and the IV. Finally, the owner can use e' and IV to decrypt the location report.

6.2 Losing

An OF device that loses its Internet connection starts emitting BLE advertisements. This advertisement consists of the 224 bit (28 bytes) public part[2] of the advertisement key (p_i), but required some engineering effort to fit in a single BLE packet.

Advertisement Packet Format. Apple had to engineer its way around the fact that one BLE advertisement packet may contain at most 37 bytes [19, Vol. 6, Part B, § 2.3.1.3], of which 6 bytes are reserved for the advertising MAC address, and up to 31 can be used for the payload. For standard compliance, the custom OF advertisements needs to add a 4-byte header for specifying *manufacturer-specific data*, which leaves 27 bytes. Within this space, Apple uses a custom encoding for subtypes used by other wireless services such as AirDrop [21]), which leaves 25 bytes for OF data. To fit the 28-byte advertisement key in one packet, Apple repurposes the random address field to encode the key's first 6 bytes. However, there is one caveat: the BLE standard requires that the first two bits of a random address be set to $0b11$. OF stores the first two bits of the advertisement key together with the 24 remaining bytes in the payload to solve the problem. We depict the complete BLE advertisement packet format in Tab. 2. Apple confirmed the reverse-engineered specification later [6].

Table 2. OF advertisement format (with zero-indexed bytes).

Bytes	Content (details cf. [6, § 5.1])
0–5	BLE address $((p_i[0] \mid (0b11 \ll 6)) \mid\mid p_i[1..5])$
6	Payload length in bytes (30)
7	Advertisement type (0xFF for manufacturer-specific data)
8–9	Company ID (0x004C)
10	OF type (0x12)
11	OF data length in bytes (25)
12	Status (e.g., battery level)
13–34	Public key bytes $p_i[6..27]$
35	Public key bits $p_i[0] \gg 6$
36	Hint (0x00 on iOS reports)

Advertising Interval. The same key is emitted during a window of 15 minutes, after which the next key p_{i+1} is used. During a window, OF-enabled iOS and macOS devices emit one BLE advertisement every two seconds when they lose Internet connectivity.

6.3 Finding

All finder devices regularly scan for OF advertisements. When the finder receives a packet in the OF advertisement format, it generates and uploads an encrypted location report to Apple's servers.

Generating Reports. The finder parses the public key from the advertisement. Then, it determines its current geolocation and creates a message that includes location, accuracy,[3] and status information (cf. green fields in Fig. 2). The message is then encrypted us-

2 More precisely, OF only advertises the X coordinate of the public key, which has a length of 28 bytes. The Y coordinate is irrelevant for calculating a shared secret via ECDH, so the sign bit for the compressed format [20] can be omitted.

3 We assume that the accuracy value is encoded in metric meters as it matches the experimentally determined positioning error of the coordinates in the location reports, as we show in § 7.

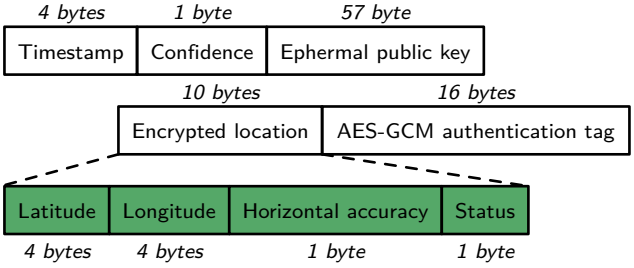

Fig. 2. Binary format of a location report.

ing the algorithm described in § 6.1. Finally, the finder creates a complete location report, including the current timestamp (in seconds since January 1, 2001), the ephemeral public key d', the encrypted message, and the AES-GCM authentication tag as shown in Fig. 2.

Uploading Reports. Finder devices accumulate reports over time and upload them in batches regularly, possibly reducing energy and bandwidth consumption. During the evaluation with our test devices, we discovered that the median time from generating to uploading a location report is 26 min. We include the delay distribution in Appendix B. The delay can increase to several hours if the finder device is in a low power mode [7]. A finder limits the number of uploaded reports for the same advertisement key to four, most likely to prevent excess traffic on Apple's servers. The upload is implemented as an HTTPS POST request to https://gateway.icloud.com/acsnservice/submit. Every request is authenticated to ensure that only genuine Apple devices can upload requests. Table 3 shows the request header containing a device identity certificate, the signing CA's certificate, and an Elliptic Curve Digital Signature Algorithm (ECDSA) signature over the request body. The certificates are stored in the device's keychain. However, the private key used for signing is stored in the Secure Enclave Processor (SEP), Apple's implementation of a trusted execution environment (TEE) [4]. The SEP prohibits the extraction of the signing key but provides an interface for signing requests. We assume that the finder authentication serves as a form of remote attestation. However, we were unable to verify this assumption due to the obfuscated code. The HTTPS request body is prefixed with a fixed header (`0x0F8AE0`) and one byte specifying the number of included reports. This limits the number of reports in a single request to 255. Each report consists the ID (SHA-256(p_i)) followed by the 88-byte location report shown in Fig. 2.

Table 3. HTTP request headers for uploading location reports.

Request Header	Value
X-Apple-Sign1	Device identity certificate (base64)
X-Apple-Sign2	SHA-256 hash of the signing CA (base64)
X-Apple-Sign3	Device ECDSA signature (ASN.1)
X-Apple-I-TimeZone	Client's time zone (e.g., GMT+9)
X-Apple-I-ClientTime	Client's time (Unix)
User-Agent	"searchpartyd/1 <iPhoneModel>/<OSVersion>"

6.4 Searching

An owner requests reported location from Apple's servers when searching for a lost device. As the advertisement keys are synchronized across all of the owner's devices, the owner can use any of their other devices with Apple's *Find My* app to download and decrypt the location reports. In short, the owner device fetches location reports from Apple's servers by sending a list of the most recent public advertisement keys of the lost device.

Downloading Reports. Similar to uploading (cf. § 6.4), downloading is implemented as an HTTPS POST request to https://gateway.icloud.com/acsnservice/fetch. We show the headers in Tab. 4 and a truncated example body in Appendix A. The user authenticates with Apple's servers using their Apple account in two steps. First, HTTP basic authentication [41] is performed with a unique identifier of the user's Apple ID[4] and a *search-party-token* that is device-specific and changes at irregular intervals (in the order of weeks). Second, several headers with so-called "anisette data" are included. Anisette data are short-lived tokens valid for 30 s and allow omitting two-factor authentication from a previously authenticated system [2].

Decrypting Reports. The response to the download request contains a list of finder location reports (cf. Fig. 2) and metadata such as the hashed public advertisement key and the time when the report was uploaded. We show a truncated example of the response body in Appendix A. Using the respective private advertisement keys d_i, the owner device can then decrypt the received location reports. Apple's *Find My* application combines a subset of the reports to display the most

4 This numerical identifier is unique to each Apple account and does not change even if the user changes their primary email address.

Table 4. HTTP request headers for downloading location reports.

Request Header	Value
Authorization	Base64 encoded basic authentication
X-Apple-I-MD	Anisette data
X-Apple-I-MD-RINFO	Anisette data
X-Apple-I-MD-M	Anisette data
X-Apple-I-TimeZone	Client's time zone
X-Apple-I-ClientTime	Client's time (ISO 8601)
X-BA-CLIENT-TIMESTAMP	Client's time (Unix)
User-Agent	"searchpartyd/1 <iPhoneModel>/<OSVersion>"

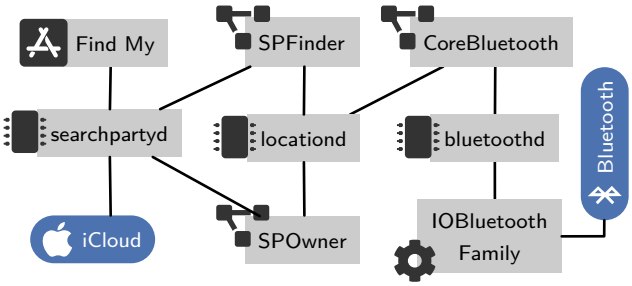

Fig. 3. Simplified view of components and their interactions such as apps (), daemons (), frameworks (), and drivers () that are used by OF on iOS. We highlight the two involved external communication interfaces in blue.

recent location of the lost device on a map. According to Apple, multiple reports are combined to get a more accurate location [4, p. 104]. While we did not reconstruct Apple's algorithm, we show in § 7 that the downloaded location reports are sufficient to not only determine the most recent location but to even precisely reconstruct and trace the movement of a lost device.

6.5 System Implementation

Apple's OF system is implemented across several daemons and frameworks which communicate via XPC, Apple's implementation of interprocess communication [12]. We depict the dependencies of the iOS implementation in Fig. 3. The main daemon that handles OF is *searchpartyd*, which runs with root privileges. It generates the necessary keys and performs all cryptographic operations. The daemon is also responsible for communicating with Apple's servers to synchronize keys, submit location reports as a finder device, and fetch location reports as an owner device. The *bluetoothd* daemon is responsible for sending and receiving OF advertisements and passes received advertisements to *locationd*. The *locationd* daemon adds the device's current location and forwards this information to *searchpartyd*, which generates the finder reports. On macOS, some functionality of *searchpartyd* such as the server communication is externalized to the *searchpartyuseragent* daemon to support the multi-user architecture that is not available on iOS.

7 Location Report Accuracy

We experimentally assess the accuracy of OF location reports submitted by finder devices. This serves two purposes. (1) From an attacker's perspective, we can determine the severity of the discovered vulnerability allowing unauthorized access to location history described in § 10. (2) From an end-user's perspective, we can provide empirical evidence on the quality of OF location reports when retrieving lost devices.

Apple's *Find My* application combines multiple location reports to improve accuracy when displaying a location on a map [4]. It does not show the seven-day history of location reports that can be available on Apple's servers. In this section, we assess the location report accuracy by using this historical location data. To this end, we use the same PoC implementation presented in § 10 to access the raw location reports for our own devices. We conduct two sets of experiments in this section. First, we evaluate the accuracy of OF reports for mobility tracking. Second, we use the seven-day location data history to profile a user's most visited or "top" locations—which can be abused for user identification. We open-source all data and evaluation tools forming this section to make our results reproducible (cf. *Availability* section).

A Note on Geographic Coordinates. Throughout this section, we deal with geographic coordinates, their distances, and their projections. In order to produce meaningful results, we need to use coordinates in the same reference system. All locations in this paper are latitude and longitude coordinates in the World Geodetic System 1984 (WGS 84) [24], which is also used by GPS. We apply the EPSG:3857 projection [25] to visualize coordinates on a map (e.g., Fig. 4). When we calculate the distance between two locations, we use the length of a geodesic, i.e., the shortest path between two points on the surface of the ellipsoidal earth [34].

7.1 Path Tracking

We compare reported OF locations with GPS traces that we record with the tracked device. We conduct experiments with different means of transportations in and around a metropolitan area. We measure the error of the

Table 5. Evaluation scenarios including distance traveled and duration of recorded GPS traces, and the number of downloaded OF location reports.

Scenario	Distance (m)	Duration (h:m:s)	No. OF reports
Walking	3375	0:55:11	489
Restaurant	160	0:42:29	185
Train	23237	0:35:30	166
Car	94569	1:04:22	25

raw OF reports to the GPS trace and show that we can significantly improve accuracy by applying a scatterplot smoothing algorithm to the raw reports.

7.1.1 Experimental Setup

We conduct experiments in different scenarios that attempt to emulate common mobility patterns: walking, driving a car, riding a train, and visiting a restaurant. All experiments were conducted in and around the city of Frankfurt am Main, Germany. For each scenario, we record a GPS trace that serves as the ground truth for our evaluation.[5]

Table 5 summarizes the evaluated scenarios with the time and distance traveled according to the GPS trace and the number of uploaded OF reports during these times. Our test devices are an iPhone 8 running iOS 13.5 and a MacBook Pro running macOS 10.15.4 that are logged into the same iCloud account. During each experiment, we carry the iPhone in flight mode to emit OF advertisements and record the GPS trace using the *SensorLog* [49] application set to a 2 s sampling interval. The MacBook acts as the owner device that we use to download location reports after each experiment. We did not carry any other Apple devices during an experiment, so we would not get additional reports. Consequently, all downloaded reports were submitted by the devices of pedestrians. To access the private advertising keys for downloading and decrypting location reports, we use our PoC implementation (cf. § 10). We use the OF reports' generation timestamps to filter the reports that lie between the start and end date of each GPS trace.

5 The experiments were conducted during the COVID-19 pandemic. Consequently, the authors implemented anti-infection measures, i.e., they wore face masks, used hand sanitizer while traveling with public transport and exercised minimum physical distancing. At the time of the experiments, the local incidence rate was low (five cases per 100 000 inhabitants over seven days) [32].

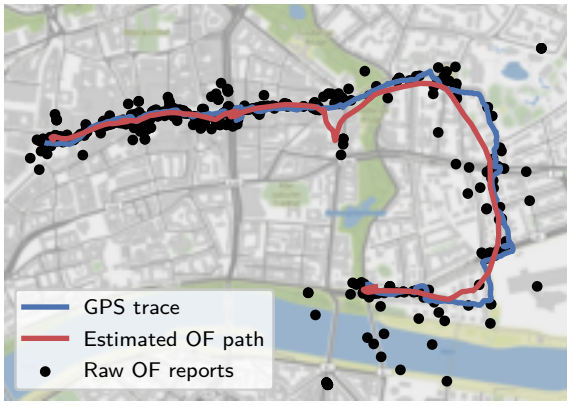

Fig. 4. Map of the *walking* scenario showing the GPS trace, the raw OF location reports, and the estimated path calculated from the reports.

Table 6. Accuracy of OF location reports.

	Mean distance to GPS trace (m)			Improvement
Scenario	Reported	Raw	Est. Path	Raw→Est. Path
Walking	121.9	81.4	25.9	3.1×
Restaurant	117.2	60.2	27.4	2.2×
Train	171.0	440.7	299.6	1.5×
Car	145.2	580.7	—	—

7.1.2 Raw Location Report Accuracy

We calculate the distance of the OF reports to the GPS trace. As GPS trace and OF reports do not have a common time index, we interpolate the GPS trace to the time indices of the OF reports. Then, we calculate the distance of each OF report to the corresponding point on the interpolated GPS trace. Table 6 shows the mean error of the raw reports. We can see that raw reports have a mean error in the order of 100 m for the *walking* and *restaurant* scenarios. However, results become less accurate when using faster modes of transportations, e.g., 500 m when riding a train. Table 6 also shows the reported accuracy value, which is part of the OF reports (cf. § 6.3), and the improved results of the estimated path that we will discuss next.

7.1.3 Estimated Path Accuracy

The raw location reports provide a decent accuracy sufficient to pinpoint an individual's location to a city district or even a street. However, when plotting the locations on a map (cf. Fig. 4), we see that simply "connecting the dots" does not yield a smooth path that we would expect from a human mobility trace. We wondered if we could improve accuracy by harnessing and

combining all finders' reports. Essentially, we want to provide a better estimate of the actual path by fitting a curve through the reported OF locations.

To this end, we apply a popular curve fitting method called locally weighted scatterplot smoothing (LOWESS) [22] independently to the longitude and latitude coordinates of the location reports. LOWESS performs a weighted local (or moving) regression over a window of nearby data points. A Gaussian distribution assigns a higher weight to the data points in the center of the window. The size of the window heavily influences the performance of LOWESS. Empirically, we found that a value of 30 reports provides the most accurate results for our traces.

Fig. 4 shows such an estimated path calculated from the *walking* scenario. The figure shows that our path estimation algorithm approximates the real path rather well. We quantify the accuracy of the estimated path by calculating the mean distance to the GPS track and show the results in Tab. 6. Most OF locations were reported in busy places in a metropolitan city (walking and restaurant visit). Here, our fitting algorithm approximated the real path with a mean error below 30 m—a 3.1× improvement over the raw data. Our fitting algorithm is unable to produce meaningful results for the *car* experiment as the sample set is too sparse. For completeness, we include maps of the restaurant, train, and car scenario in Appendix C.

Overall, using a fitting algorithm helps to reconstruct the user's path from noisy OF reports *if* the dataset is dense enough. In the best case, we improved the accuracy compared to the raw location reports by a factor of 3.1×.

7.2 Identifying Top Locations

Apple's servers store OF reports for seven days, which can be accessed by an unauthorized third-party application (cf. § 10). Previous work has shown that the top (most visited) locations can be used for user fingerprinting. Montjoye et al. [38] demonstrated that four spatio-temporal points are sufficient to identify 95 % of all individuals in an anonymized location dataset. Also, Zang and Bolot [52] found that the three top locations on a mobile cell-level are accurate enough to identify 50 % of all individuals in a large data set. Even though most devices try to keep a permanent connection to the Internet, our analysis has shown that hundreds of reports have been generated throughout a week. This section shows that OF reports can also be used to determine top locations and even achieve an accuracy of up to 5 m.

7.2.1 Experimental Setup

To determine the top locations, we want to use the same seven-day historic data available as an attacker getting access to the OF report decryption keys (cf. § 10). Collecting this data from various test subjects for an evaluation would be extremely intrusive to their privacy. Running our evaluation algorithm on the subjects' Macs would also not be feasible, because the subjects would have to disable macOS's SIP to download raw OF location reports, effectively weakening their system security. Sensibly, our ethical review board (ERB) would not approve such a study. Consequently, we abstain from conducting a user study and, instead, use our (the authors') data for the evaluation. To preserve the authors' privacy, we will not show or discuss the raw data. Instead, we apply our algorithm for identifying top locations on the raw data and then discuss the output.

7.2.2 Resampling and Clustering OF Reports

Previous works [38, 52] have used identifiers of cell sites to rank top locations. For a particular user, they simply count the number of registrations per cell and base the location rank on this number. E.g., the cell with the highest registration count is regarded as the top 1 location. We cannot apply the same concept for identifying top locations based on OF reports for two reasons.

Problem 1: Non-Uniform Distribution Over Time. We observed that the density of OF reports over time is non-uniform. In busy places such as malls or restaurants, more finder devices are available and, consequently, more reports are generated. If we simply count the number of OF reports to determine the top locations, these busy places are likely to be overrepresented. Instead, we want to rank top locations based on the overall visit duration of the user.

Problem 2: Continuous Coordinate Range. Cell identifiers are elements of a discrete set. In contrast, the location coordinates in OF reports are drawn from a continuous range, which makes simple counting of visits per location hard. To address both issues, we use a two-step approach.

Solution 1: Resampling. We resample the location reports on the time axis to solve Problem 1 and "flatten" the distribution over time. Within each resampling interval R, we calculate the center coordinates as the mean of all reports within that interval. Setting a reasonable value R is essential. If R is too small, the desired

Table 7. Identified top locations sorted by their rank. Error indicates the geodesic distance between the ground truth and the estimated cluster center. Days indicate on how many days of the week the location was visted. Dwell time is the estimated overall time that the location was visited.

Kind	Rank	Error	Resampled reports	Days	Dwell time (hour:min)
Home	1	14.1 m	129	6	43:00
Work	2	4.9 m	25	2	08:20
Partner	3	15.5 m	19	2	06:20
Friends	4	11.1 m	15	1	05:00
Sport	5	9.5 m	9	3	03:00
Family	6	6.6 m	9	2	03:00

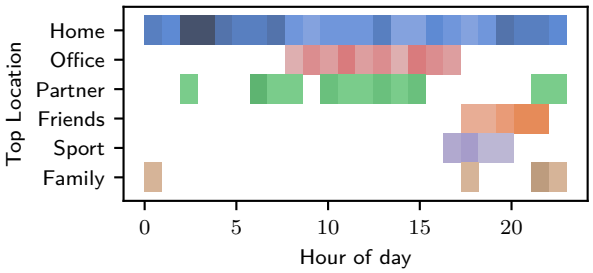

Fig. 5. Visiting times distribution of top locations averaged over seven days. Darker colors indicate a higher number of location reports during these hours.

flattening effect will not occur. If R is too large, we lose accuracy. Empirically, we found that $R = 20\,\text{min}$ produces good results for our dataset.

Solution 2: Clustering. We use a clustering algorithm to identify places for which we received multiple location reports over time to solve Problem 2. In particular, we select the popular Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [26, 43] algorithm. DBSCAN can detect an arbitrary number of clusters, which is important for us as we do not know the number of top locations a priori. Also, DBSCAN adequately deals with noise, which is common in OF reports. In short, DBSCAN forms clusters by finding "core samples" that have at least N neighbors within a radius of D. All samples that are not part of a cluster are considered as noise. Empirically, we determined that $D = 50\,\text{m}$ and $N = 6$ produces good results for our dataset.

7.2.3 Results

We run our resampling and clustering algorithms on an author dataset. Together, the algorithms determine a list of six clusters that we interpret as top locations. Afterwards, we let the author label each entry and assign the actual location of the home or office by picking coordinates from an online map service, which we use as ground truth. Finally, we compute the distance of the clusters' centers to the ground truth locations and show the complete results in Tab. 7. The results demonstrate that we successfully identified typical locations such as home and workplace [52] with an error below 20 m. Impressively, the location reports were precise enough to pinpoint not only the workplace but with a 5 m error even identify the exact office.

We wondered if it would have been possible to discern the type of location (e.g., home or work) just from the characteristics of the location reports. To this end, we provide a detailed view of the visiting times across the hours of a day in Fig. 5. The figure shows the hours of a day in which location reports were submitted for the respective top location. We can visually identify the type of some locations from this distribution. In particular, the workplace distribution matches typical office hours (8 am to 5 pm). The all-day distribution of the home location reflects the fact that the measurements were taken over the course of a week and the author stayed at home on some days for remote work.

Previous work showed that four spatio-temporal points could be sufficient to uniquely identify an individual with a chance of 95 % [38]. While we were unable to conduct a similar large-scale user study as in [38] based on OF, our small-scale experiment already demonstrates that OF reports contain highly sensitive information that can be used to deanonymize users.

8 Security and Privacy Analysis

In this section, we perform a security and privacy analysis of Apple's OF system implemented on iOS and macOS based on the adversary models described in § 4. We first examine the cryptography-related components that are relevant for the local application **(A1)** and service operator **(A4)** models that have access to keys and encrypted reports, respectively. Then, we assess the BLE interface relevant to the proximity-based adversary **(A2)** and the HTTPS-based server communication relevant for the network-based adversary **(A3)**. We summarize our findings in Tab. 8 and discuss in the following.

8.1 Cryptography

Key Diversification. OF employs key diversification to derive the rolling advertisement keys from the master beacon key (cf. § 6.1). Apple's design follows the

Table 8. Summary of our security and privacy analysis of OF. We list potential issues and provide a short assessment about whether or not they are exploitable in practice (✓or ✗) and if ✓, by which adversary model (cf. § 4).

Component	Potential issue	Exploit.	Assessment
Cryptography	Key diversification	✗	The custom key diversification process follows the NIST recommendation for key derivation through extraction-then-expansion [16].
	Choice of P-224 curve	✗	Use of NIST P-224 is discouraged by some cryptographers [18]. However, we are unaware of any practical attacks against P-224 when used exclusively for ECDH.
	Insecure key storage	✓ **(A1)**	Keychains and SEP are used to securely store keys for server communication and the master beacon key. However, macOS caches the derived advertisement keys on disk, which can be read by local applications. Attackers can exploit this to access (historical) geolocation data as we describe in § 10.
Bluetooth	Device tracking via BLE advertisements	✗	BLE payload and address are determined by the advertisement key, which is changed at 15 min intervals, making long-term tracking hard.
	Remote code execution (RCE)	✗	As OF uses non-connectable mode to emit advertisements, devices remain secure against RCE attacks on the Bluetooth firmware [42].
	Denial-of-service (DoS)	✓ **(A2)**	An attacker could emit/relay legitimate advertisements at other physical locations to pollute the set of location reports.
Server comm.	Spoofing (finder)	✗	Impact similar to Bluetooth relaying. However, we have been unable to inject fabricated location reports into the server communication.
	Spoofing (owner)	✗	Spoofing an owner device is not critical as location reports are end-to-end encrypted.
	Device identification	✓ **(A4)**	Apple's servers can identify both finder and owner devices. This enables a location correlation attack that we discuss in § 9.

NIST recommendation of performing extraction-then-expansion [16] to securely derive keys. The two-step process first extracts a derivation key from a secure input and then expands this key to the desired output length. Specifically, OF first extracts a new 32-byte key SK_i from the previous derivation key using the KDF and then expands SK_i using the same KDF to 72 bytes.

Choice of NIST P-224 Curve. We believe that Apple's choice for the NIST P-224 curve is the consequence of the constrained capacity of BLE advertisements while maximizing the security level of the encryption keys. Apple's implementation of P-224 in *corecrypto* has been submitted to validate compliance with U.S. Federal Information Processing Standards (FIPS) [9]. Within the cryptography community, some researchers discourage the use of P-224 because its generation process is unclear [17, 18]. More modern curves with the same security margin are available, e.g., M-221 [13], but are not used by Apple.

Insecure Key Storage. We analyzed how OF keys and secrets are stored on the system. While most involved keys are synchronized and stored in the iCloud keychain, we discovered that the advertisement keys derived from the master beacon key (cf. § 6.1) are cached on disk to avoid unnecessary re-computations. We found that the cached key directory is accessible by a local application with user privileges and can be used to bypass the system's location API, as we describe in § 10.

8.2 Bluetooth

Device Tracking. One of the key design goals of OF is to prevent tracking of lost devices via their BLE advertisements. According to our analysis, OF fulfills this promise by randomizing both BLE advertisement address and payload in 15 min intervals (cf. § 6.2).

Remote Code Execution. In addition, OF uses the so-called "non-connectable mode" [19, Vol. 3, Part C, § 9.3.2], which means that other devices cannot connect to it and exploit potential remote code execution (RCE) vulnerabilities in the Bluetooth firmware [42].

Denial-of-Service Through Relaying. BLE advertisements only contain the public part of an advertisement key and are not authenticated. Anyone recording an advertisement can replay it at a different physical location. Any finder at that location would generate a location report and submit it to Apple. Through this type of relaying, an attacker could make a lost device appear at a different location, effectively mounting a DoS attack as owners would receive different contradicting location reports.

8.3 Server Communication

Spoofing. The communication with Apple's servers uses TLS, including certificate pinning to ensure that no MitM attack can be deployed. Based on our analysis, the protocol seems to implement a secure authentication

scheme. However, we have been unable to reconstruct some of the involved components. We understand that a device-specific certificate (cf. § 6.3) and a private signing key, protected by the SEP, are involved in submitting reports. We *assume* that this private key is used for remote attestation to prevent non-Apple devices from submitting potentially fabricated reports. The generation and registration process of these keys with Apple's server remains unknown to us. Also, the "anisette data" used for authenticating owner devices (cf. § 6.4) is not publicly documented, and the code that generates the tokens is highly obfuscated.

Device Identification. While we did not recover the exact details of the authentication mechanism, we have observed that both finder and owner devices provide identifiable tokens to Apple's servers. In particular, owner devices provide their Apple ID to access location reports. In § 9, we show that by requesting IDs, Apple's servers are—in principle—able to correlate the locations of different owners.

9 Apple Can Correlate User Locations

Apple as the service provider **(A4)** could infer that two or more owners have been in close proximity to each other as OF uses identifiable information in both upload and download requests. Law enforcement agencies could exploit this issue to deanonymize participants of (political) demonstrations even when participants put their phones in flight mode. Exploiting this design vulnerability requires that the victims request the location of their devices via the Find My application.[6] Next, we describe the vulnerability, a possible attack, and our proposed mitigation.

9.1 Vulnerability

When uploading and downloading location reports, finder and owner devices reveal their identity to Apple. During the upload process, the finder reveals a device-specific identifier in the HTTPS request header (cf. Tab. 3) that can be used to link multiple reports to the same finder. Similarly, during the download process, the owner device has to reveal its Apple ID. In particular, the owner includes its Apple ID in the HTTPS request headers (cf. Tab. 4), which allows Apple to link reports uploaded by a particular finder to the Apple ID of the downloading owners. Since we do not have access to Apple's servers, we cannot make assumptions about whether or not Apple actually stores such metadata. However, the fact that Apple *could* store this information indefinitely opens the possibility of abuse.

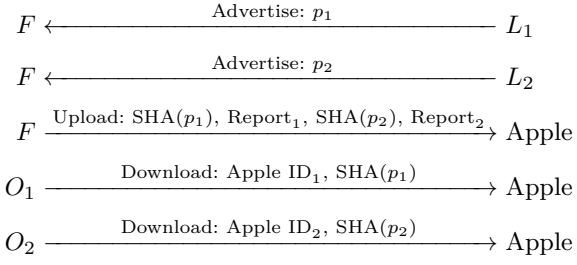

Fig. 6. Apple could infer which users have been in close proximity to each other.

9.2 Attack

It is possible for Apple to find out which owners have been in physical proximity to each other *if the owners request the location of their devices via the Find My application.* We sketch the attack for two owners in Fig. 6. A finder F receives advertisements from the lost devices L_1 and L_2 that belong to the owners O_1 and O_2, respectively, and publishes encrypted location reports to Apple's servers. Due to the limited communication range of BLE, we can reasonably assume that L_1 and L_2 have been in close proximity if the respective location reports were generated at a similar time and submitted by the same finder. Later, O_1 and O_2 both download location reports, by opening the *Find My* app, for L_1 and L_2, respectively. At this point, Apple can infer that these two owners identified by their Apple IDs were close to each other.

9.3 Impact

The presented attack could be harmful to protesters who put their phones into flight mode to stay anonymous and prevent their devices from showing up during a cell site analysis—which is precisely when the devices would start emitting OF advertisements. Law enforcement agencies could record all the advertised public keys at the demonstration site and ask Apple to provide the

6 This requirement currently limits the exploitability of the described vulnerability. However, exploitability could increase when changing the usage pattern of the downloading API (cf. § 6.4). For example, a future Apple application could notify users about lost devices automatically by regularly requesting reports for the devices' last known locations, thereby, removing the need for user interaction.

Apple IDs of the users that later requested location reports to deanonymize the participants. Such a collusion would be a combination of the proximity-based **(A2)** and service provider **(A4)** adversary models (cf. § 4).

9.4 Proposed Mitigation

There are two straightforward options to mitigate this attack: remove identifying information from either (1) finder devices or (2) owner devices. We assume that the authentication of the finder provides a form a remote attestation proving that the device is—in fact—a genuine Apple device allowed to upload location reports to Apple's servers. In that case, option (1) is not feasible as the finder has to provide some verifiable information by design. However, we currently see no reason why owner devices have to authenticate to Apple's servers and provide personally identifiable information, i.e., the Apple ID. We found that any Apple device can request arbitrary location reports, so the authentication appears to be a security-by-obscurity measure and only prevents everyone without access to an Apple device from accessing location reports. Therefore, we recommend option (2) as mitigation and disable authentication for download requests.

10 Unauthorized Access of Location History

We discovered a vulnerability of the OF implementation on macOS that allows a malicious application **(A1)** to effectively circumvent Apple's restricted location API [5] and access the geolocation of all owner devices without user consent. Moreover, historical location reports can be abused to generate a unique mobility profile and identify the user, as we demonstrate in § 7.

10.1 Vulnerability

§ 6 describes that the location privacy of lost devices is based on the assumption that the private part of the advertisement keys is only known to the owner devices. The advertisement keys change every 15 minutes and OF supports retrieving location reports from the last seven days, so there is a total of 672 advertisement keys per device, for which there exist potential location reports on Apple's servers. In principle, all of these keys could be generated from the master beacon key (cf. § 6.1) whenever needed. However, Apple decided to cache the advertisement keys, most likely for performance reasons. During our reverse engineering efforts, we found that macOS stores these cached keys on

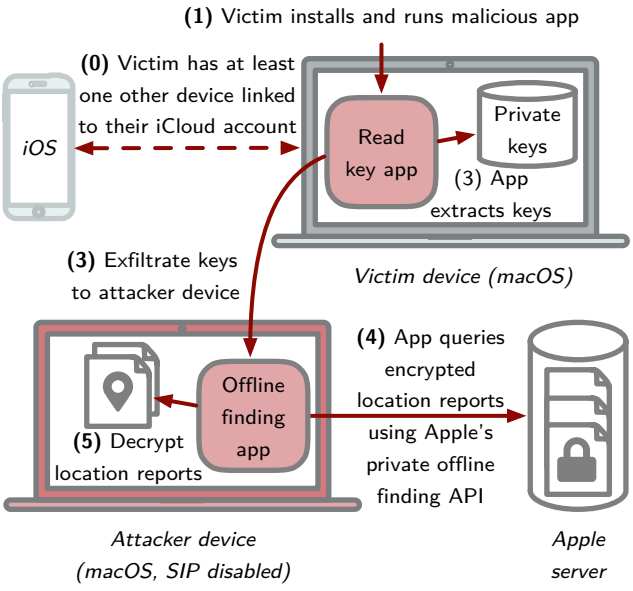

Fig. 7. Attack flow for gaining access to the victim's location history. Attacker-controlled components are marked in red.

disk in the directory `/private/var/folders/<Random>/com.apple.icloud.searchpartyd/Keys/<DeviceId>/Primary/<IdRange>.keys`. The directory is readable by the local user and—in extension—by any application that runs with user privileges. On iOS, those cache files exist as well, but they are inaccessible for third-party applications due to iOS's sandboxing mechanism.

10.2 Attack

We describe the attack flow and explain our PoC implementation, which leads to the attacker gaining access to the location history of the victim's devices. In the following, we detail the operation of our two-part PoC attack. The steps are referring to Fig. 7.

Reading Private Keys (Steps 1–3). The victim installs a non-sandboxed malicious application.[7] When started, the malicious application runs with user privileges and, therefore, has access to the key cache directory. It can read the advertisement keys from disk (2) and then exfiltrate them to the attacker's server (3). Apart from starting the application, this process requires no user interaction, i.e., no dialogs requesting disk access are displayed to the user.

7 Sandboxing is only required for applications distributed via Apple's app store. Many popular macOS applications such as Firefox or Zoom are not distributed via the app store and, thus, could have exploited the discovered vulnerability.

Downloading Location Reports (Step 4). The *attacker's machine* essentially acts as an owner device (cf. § 6.4) but uses the victim's keys as input for the HTTPS download request. To download the victim's location reports, our PoC needs to access the attacker's *anisette data* for authenticating the request to Apple's servers. As we need to link private frameworks and access the anisette data in our implementation, the attacker's macOS system needs to disable SIP and Apple mobile file integrity (AMFI). Since this device is attacker-owned, this requirement does not limit the applicability of the presented attack. SIP and AMFI are disabled by booting in the macOS recovery mode and running the following terminal commands.

```
csrutil disable
nvram boot-args="amfi_get_out_of_my_way=1"
```

Decrypting Location Reports (Step 5). In the final step, the adversary uses the victim's private keys to decrypt the location reports.

10.3 Impact

The attack essentially allows any third-party application to *bypass Apple's Core Location API* [5] that enforces user consent before an application can access the device's location. Moreover, the attacker can access the location history of the past seven days of *all* the owner's devices. The victim is only required to download and run the application but remains otherwise clueless about the breach. Our analysis has shown that the advertisement keys are precomputed for up to *nine* weeks into the future, which allows an adversary to continue downloading new reports even after the victim has uninstalled the malicious application.

Even though the location reports are not continuous, our evaluation in § 7 shows that it is easy to identify the user's most visited places such as home and workplace. Furthermore, we show that the decrypted location reports can accurately track the victim's movement of the last seven days.

10.4 Mitigation

As a short-term mitigation, users can disable participating in the OF network to prevent the attack. In addition, we propose three long-term solutions to mitigate the attack: (1) encrypting all cached files on disk store the decryption key in the keychain, (2) restricting access to the cache directory via access control lists, (3) not caching the keys and computing them on-demand. In fact, macOS 10.15.7 includes a mitigation based on option (2), which moved the keys to a new directory that is protected via the system's sandboxing mechanism.

11 Related Work

We review other crowd-sourced location tracking systems and previous security and privacy analyses of Apple's ecosystem.

Crowd-Sourced Location Tracking. Weller et al. [51] have studied the security and privacy of commercial Bluetooth tags (similar to Apple's definition of *accessories*) sold by multiple vendors. Many of the studied systems provide crowd-sourced location tracking similar to Apple's OF, allowing users to discover lost devices by leveraging the finder capabilities of other devices. The study discovered several design and implementation issues, including but not limited to the use of plaintext location reports, unauthorized access to location reports, broken TLS implementations, and leaking user data. Based on their findings, Weller et al. [51] propose a novel privacy-preserving crowd-sourced location tracking system called *PrivateFind*. PrivateFind does not need user accounts and uses end-to-end encrypted location reports with a symmetric encryption key stored on the Bluetooth finder during the initial setup. In their solution, a finder that discovers a lost Bluetooth tag sends its geolocation to the finder over Bluetooth. The lost device encrypts the location with its symmetric key and returns the encrypted report. The finder then uploads the encrypted location report on behalf of the tag. An owner device that knows the symmetric key can then download and decrypt the location report.

To the best of our knowledge, PrivateFind is the only other privacy-friendly offline device finding system next to OF. Both designs achieve similar privacy goals, such as preventing a third party from learning the location. The main difference is the way encrypted location reports are generated. OF employs public-key cryptography, which allows finder devices to generate a location report upon receiving a single Bluetooth advertisement. In PrivateFind, lost devices are actively involved in the generation, which leads to the following practical issues: (1) Lost devices or tags drain their batteries quicker as they have to establish Bluetooth connections with other devices and perform cryptographic operations. This opens up the door for resource-exhaustion attacks where a powerful adversary issues an excessive number of encryption requests to the lost devices. (2) The lack of finder attestation would allow an attacker to upload fabricated reports as the lost device cannot verify the correctness of the provided location.

Apple's Wireless Ecosystem Security and Privacy. Previous work analyzed parts of Apple's wireless services. Bai et al. [15] investigated the risks of using insecure multicast DNS (mDNS) service advertisements and showed that they have been able to spoof an AirDrop receiver identity to get unauthorized access to personal files. Stute, Kreitschmann, and Hollick [46] and Stute et al. [48] reverse engineered the complete AWDL and AirDrop protocols and demonstrated several attacks, including user tracking via AWDL, a DoS attack on AWDL, and a MitM attack on AirDrop. Martin et al. [36] extensively analyzed the content of the BLE advertisements for several Apple services. They found several privacy-compromising issues, including device fingerprinting and long-term device and activity tracking. Celosia and Cunche [21] extended this work and discovered new ways of tracking BLE devices such as Apple AirPods and demonstrated how to recover user email addresses and phone numbers from BLE advertisements sent by Apple's Wi-Fi Password Sharing (PWS). Heinrich et al. [30] found that AirDrop also leaks user phone numbers and email addresses and proposes a new protocol based on private set intersection. Stute et al. [45] investigated the protocols involved in PWS and Apple's Handoff and found vulnerabilities allowing device tracking via Handoff advertisements, a MitM attack on PWS, and DoS attacks on both services. While the above works have analyzed other services, we leveraged their methodology for approaching our analysis and reverse engineering work of OF.

12 Conclusion

Apple has turned its mobile ecosystem into a massive crowd-sourced location tracking system called OF. In this system, all iPhones act as so-called finder devices that report the location of lost devices to their respective owners. Apple claims to implement OF in a privacy-preserving manner. In particular, location reports are inaccessible to Apple, finder identities are concealed, and BLE advertisements cannot be used to track the owner [35]. We have been the first to challenge these claims and provide a comprehensive security and privacy analysis of OF.

The good news is that we were unable to falsify Apple's specific claims. However, we have found that OF provides a critical attack surface that seems to have been outside of Apple's threat model. Firstly, the OF implementation on macOS allows a malicious application to effectively bypass Apple's location API and retrieve the user's location without their consent. By leveraging the historical reports, an attacker is able to identify the user's most visited location with sub-20 m accuracy. Secondly, we believe that Apple has yet to provide a good reason why owner devices need to authenticate when retrieving encrypted location reports as it allows Apple to correlate the locations of different Apple users.

We were only able to publish our findings by intensively studying the OF system using reverse engineering, which is a very time-consuming process (we started analyzing OF mid-2019). To protect user privacy, we believe that systems handling highly sensitive information such as OF need to be *openly and fully* specified to facilitate *timely* independent analyses. To this end, we urge manufacturers to provide not only partial [6] but complete documentation of their systems and release components as open-source software whenever possible, which is already a best practice for cryptographic libraries [9].

Responsible Disclosure

We disclosed the vulnerability in § 10 on July 2, 2020. On October 5, 2020, Apple informed us that macOS 10.15.7 provides a mitigation for the issue, which was assigned CVE-2020-9986. In addition, we informed Apple about the vulnerability in § 9 on October 16, 2020, and are currently waiting for feedback.

Availability

We release the following open-source software artifacts as part of the Open Wireless Link project [47]: (1) The PoC implementation that can download and decrypt location reports, which we used for the exploit described in § 10 (github.com/seemoo-lab/openhaystack). (2) The experimental raw data and evaluation scripts to reproduce the results in § 7 (github.com/seemoo-lab/offline-finding-evaluation).

Acknowledgments

We thank our anonymous reviewers and our shepherd Santiago Torres-Arias for their invaluable feedback. We thank Fontawesome for the vector graphics and Stamen for the map tiles used in our figures. This work has been funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[1] Oleg Afonin. *Extracting and Decrypting iOS Keychain: Physical, Logical and Cloud Options Explored*. Elcomsoft Co. Ltd. 2020. URL: https://blog.elcomsoft.com/2020/08/extracting-and-decrypting-ios-keychain-physical-logical-and-cloud-options-explored/ (visited on 02/08/2021).

[2] Oleg Afonin. *iCloud Authentication Tokens Inside Out*. Elcomsoft Co. Ltd. 2017. URL: https://blog.elcomsoft.com/2017/11/icloud-authentication-tokens-inside-out (visited on 09/03/2020).

[3] Apple Inc. *App Review*. URL: https://developer.apple.com/app-store/review/ (visited on 02/09/2021).

[4] Apple Inc. *Apple Platform Security*. 2020. URL: https://support.apple.com/guide/security/ (visited on 10/10/2020).

[5] Apple Inc. *Core Location*. URL: https://developer.apple.com/documentation/corelocation/ (visited on 10/10/2020).

[6] Apple Inc. *Find My Network Accessory Specification*. Version Release R1. 2020. URL: https://developer.apple.com/find-my/.

[7] Apple Inc. *Maximizing Battery Life and Lifespan*. 2020. URL: https://www.apple.com/batteries/maximizing-performance/ (visited on 10/07/2020).

[8] Apple Inc. *Notarizing macOS Software Before Distribution*. URL: https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution (visited on 11/24/2020).

[9] Apple Inc. *Security*. URL: https://developer.apple.com/security/ (visited on 09/16/2020).

[10] Apple Inc. *WWDC 2019 Keynote*. 2019. URL: https://developer.apple.com/videos/play/wwdc2019/101/ (visited on 08/17/2020).

[11] Apple Inc. *WWDC 2020 Keynote*. 2020. URL: https://developer.apple.com/videos/play/wwdc2020/101/ (visited on 08/17/2020).

[12] Apple Inc. *XPC*. URL: https://developer.apple.com/documentation/xpc (visited on 09/03/2020).

[13] Diego F. Aranha, Paulo S. L. M. Barreto, Geovandro C. C. F. Pereira, and Jefferson E. Ricardini. "A Note on High-Security General-Purpose Elliptic Curves." In: *Cryptology ePrint Archive* (2013). URL: https://eprint.iacr.org/2013/647.

[14] Ethan Arbuckle. *Unredacting Private os_log Messages on iOS*. 2018. URL: https://github.com/EthanArbuckle/unredact-private-os_logs (visited on 02/10/2021).

[15] Xiaolong Bai, Luyi Xing, Nan Zhang, Xiaofeng Wang, Xiaojing Liao, Tongxin Li, and Shi-Min Hu. "Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf." In: *IEEE Symposium on Security and Privacy (S&P)*. 2016. DOI: 10.1109/SP.2016.45.

[16] Elaine Barker, Lily Chen, and Richard Davis. *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*. Special Publication 800-56C Rev. 1. 2018. DOI: 10.6028/nist.sp.800-56cr1.

[17] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records." In: *Public Key Cryptography - PKC 2006*. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11745853_14.

[18] Daniel J. Bernstein and Tanja Lange. *SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. 2020. URL: https://safecurves.cr.yp.to (visited on 10/07/2020).

[19] Bluetooth SIG. *Bluetooth Core Specification Version 5.2*. Tech. rep. 2019.

[20] Daniel R. L. Brown. *Standards for Efficient Cryptography 1 (SEC 1)*. 2009.

[21] Guillaume Celosia and Mathieu Cunche. "Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols." In: *Privacy Enhancing Technologies* (2020). DOI: 10.2478/popets-2020-0003.

[22] William S. Cleveland and Susan J. Devlin. "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting." In: *Journal of the American Statistical Association* 83.403 (1988). DOI: 10.1080/01621459.1988.10478639.

[23] Quang Do, Ben Martini, and Kim-Kwang Raymond Choo. "The Role of the Adversary Model in Applied Security Research." In: *Computers & Security* 81 (2019). DOI: 10.1016/j.cose.2018.12.002.

[24] EPSG Geodetic Parameter Dataset. *WGS 84 (EPSG:4326)*. URL: https://epsg.org/crs_4326/WGS-84.html (visited on 10/13/2020).

[25] EPSG Geodetic Parameter Dataset. *WGS 84 / Pseudo-Mercator (EPSG:3857)*. URL: https://epsg.org/crs_3857/WGS-84-Pseudo-Mercator.html (visited on 10/13/2020).

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *International Conference on Knowledge Discovery and Data Mining*. KDD-96. AAAI Press, 1996. URL: http://www.aaai.org/Library/KDD/1996/kdd96-037.php.

[27] George Garside. *Show Private Log Messages in Catalina's Console.app*. 2020. URL: https://georgegarside.com/blog/macos/sierra-console-private/ (visited on 09/15/2020).

[28] Matthew Green. *How does Apple (privately) find your offline devices?* 2019. URL: https://blog.cryptographyengineering.com/2019/06/05/how-does-apple-privately-find-your-offline-devices/ (visited on 09/17/2020).

[29] Andy Greenberg. *The Clever Cryptography Behind Apple's 'Find My' Feature*. 2019. URL: https://www.wired.com/story/apple-find-my-cryptography-bluetooth/ (visited on 09/17/2020).

[30] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. "PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop." In: *USENIX Security Symposium*. To appear. USENIX Association, 2021.

[31] Alexander Heinrich, Milan Stute, and Matthias Hollick. "BTLEmap: Nmap for Bluetooth Low Energy." In: *Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2020. DOI: 10.1145/3395351.3401796.

[32] Hessisches Landesprüfungs- und Untersuchungsamt im Gesundheitswesen. *Bulletin Stand 29.07.2020, 14 Uhr*. 2020. URL: https://soziales.hessen.de/sites/default/files/media/2020_07_29_bulletin_coronavirus.pdf (visited on 11/24/2020).

[33] American National Standards Institute. *ANSI X.963 Public-Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*. Tech. rep. 2001.

[34] Charles F. F. Karney. "Algorithms for Geodesics." In: *Journal of Geodesy* 87 (2013). DOI: 10.1007/s00190-012-0578-z.

[35] Ivan Krstić. "Behind the Scenes of iOS and Mac Security." In: *Black Hat USA 2019*. 2019. URL: https://www.youtube.com/watch?v=3byNNUReyvE&t=2398s (visited on 09/09/2020).

[36] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik Rye, Brandon Sipes, and Sam Teplov. "Handoff All Your Privacy: A Review of Apple's Bluetooth Low Energy Implementation." In: (2019). DOI: 10.2478/popets-2019-0057.

[37] David A. McGrew, Kevin M. Igoe, and Margaret Salter. *Fundamental Elliptic Curve Cryptography Algorithms*. RFC 6090. IETF, 2011. DOI: 10.17487/RFC6090.

[38] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. "Unique in the Crowd: The Privacy Bounds of Human Mobility." In: *Scientific Reports* 3.1 (2013). DOI: 10.1038/srep01376.

[39] National Institute for Standards and Technology. *Digital Signature Standard*. 186-2. 2000. URL: http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[40] Ole André V. Ravnås. *Frida: A World-Class Dynamic Instrumentation Framework*. 2020. URL: https://frida.re (visited on 09/23/2020).

[41] Julian F. Reschke. *The 'Basic' HTTP Authentication Scheme*. RFC 7617. IETF, 2015. DOI: 10.17487/RFC7617.

[42] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. "Frankenstein: Advanced Wireless Fuzzing to Exploit New Bluetooth Escalation Targets." In: *USENIX Security Symposium*. USENIX Association, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/ruge.

[43] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN." In: *ACM Transactions on Database Systems* 42.3 (2017). DOI: 10.1145/3068335.

[44] Milan Stute. "Availability by Design: Practical Denial-of-Service-Resilient Distributed Wireless Networks." PhD thesis. 2020. DOI: 10.25534/tuprints-00011457.

[45] Milan Stute, Alexander Heinrich, Jannik Lorenz, and Matthias Hollick. "Disrupting Continuity of Apple's Wireless Ecosystem Security: New Tracking, DoS, and MitM Attacks on iOS and macOS Through Bluetooth Low Energy, AWDL, and Wi-Fi." In: *USENIX Security Symposium*. To appear. USENIX Association, 2021.

[46] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the Apple Wireless Direct Link Ad hoc Protocol." In: *International Conference on Mobile Computing and Networking*. ACM, 2018. DOI: 10.1145/3241539.3241566.

[47] Milan Stute, David Kreitschmann, and Matthias Hollick. *The Open Wireless Link Project*. 2018. URL: https://owlink.org.

[48] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. "A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link." In: *USENIX Security Symposium*. USENIX Association, 2019. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/stute.

[49] Bernd Thomas. *SensorLog*. 2020. URL: https://apps.apple.com/us/app/sensorlog/id388014573 (visited on 09/04/2020).

[50] Nghia Tran and Hang Nguyen. *Proxyman*. URL: https://proxyman.io (visited on 09/15/2020).

[51] Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. "Lost and Found: Stopping Bluetooth Finders from Leaking Private Information." In: *Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2020. DOI: 10.1145/3395351.3399422.

[52] Hui Zang and Jean Bolot. "Anonymization of Location Data Does Not Work: A Large-Scale Measurement Study." In: *International Conference on Mobile Computing and Networking*. ACM, 2011. DOI: 10.1145/2030613.2030630.

A HTTP Body Content

Listings 1 and 2 show the HTTP request and response body for downloading location reports, respectively.

Listing 1. Request body for downloading location reports.

```
{
  "search": [
    {
      "endDate": 1599052814928,
      "startDate": 1598965514928,
      "ids": [
        "tEJGn1j59g+mgj7cKhDMYN3UMNb8...",
        "sr74jRoVkhXdshf0Y68j6qGyW68v...",
        "pzcyP8dXfdSyVTHk8io7AUgAx85J...",
        ... ]
    }, ... ]
}
```

Listing 2. Response body for downloading location reports.

```
{
  "results": [
    {
      "datePublished": 1586804587284,
      "payload": "JETtmwIEzRBG....",
      "description": "found",
      "id": "B6E5tpUPbuudAc..."
      "statusCode": 0
    }, ... ],
  "statusCode": "200"
}
```

B Reporting Delay

Fig. 8 shows the distribution of the reporting delays (time between uploading and generating a report) over all traces recorded for the experiments in § 7.1.

C Additional Experimental Traces

We show the reports of our *restaurant*, *train*, and *car* evaluation scenarios in Figs. 9 to 11, respectively.

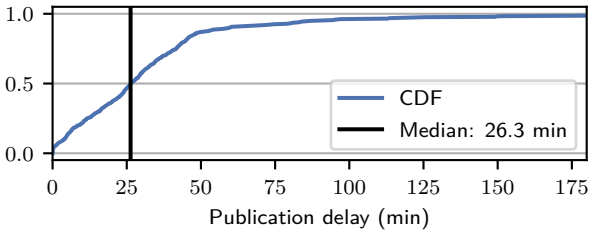

Fig. 8. Reporting delays for all reports considered in § 7.1 as a cumulative distribution function.

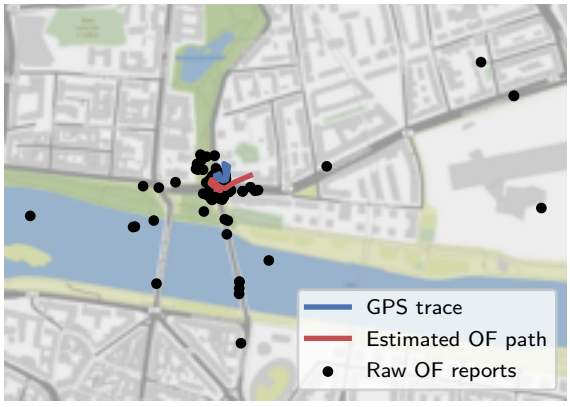

Fig. 9. Map showing the GPS trace, the raw OF location reports, and the estimated paths for the *restaurant* scenario (cf. § 7.1).

Fig. 10. Map showing the GPS trace, the raw OF location reports, and the estimated paths for the *train* scenario (cf. § 7.1).

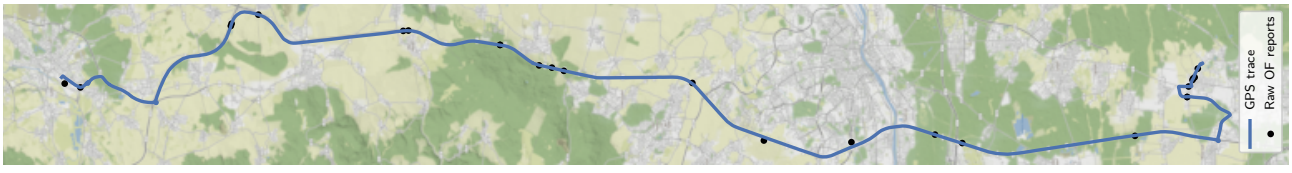

Fig. 11. Map showing the GPS trace and the raw OF location reports for the *car* scenario (cf. § 7.1).