Security and Privacy Analysis of Samsung's Crowd-Sourced Bluetooth Location Tracking System

Tingfeng Yu, James Henderson, Alwen Tiu, and Thomas Haines,
School of Computing, The Australian National University

https://www.usenix.org/conference/usenixsecurity24/presentation/yu-tingfeng

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the
33rd USENIX Security Symposium
is sponsored by USENIX.

Security and Privacy Analysis of Samsung's Crowd-Sourced Bluetooth Location Tracking System

Tingfeng Yu
School of Computing
The Australian National University

James Henderson
School of Computing
The Australian National University

Alwen Tiu
School of Computing
The Australian National University

Thomas Haines
School of Computing
The Australian National University

Abstract

We present a detailed analysis of Samsung's Offline Finding (OF) protocol, which is part of Samsung's Find My Mobile system for locating Samsung mobile devices and Galaxy SmartTags. The OF protocol uses Bluetooth Low Energy (BLE) to broadcast a unique beacon for a lost device. This beacon is then picked up by nearby Samsung phones or tablets (the *helper* devices), which then forward the beacon and the location it was detected at, to a vendor server. The owner of a lost device can then query the server to locate their device. We examine several security and privacy related properties of the OF protocol and its implementation. These include: the feasibility of tracking an OF device through its BLE data, the feasibility of unwanted tracking of a person by exploiting the OF network, the feasibility for the vendor to de-anonymise location reports to determine the locations of the owner or the helper devices, and the feasibility for an attacker to compromise the integrity of the location reports. Our findings suggest that there are privacy risks on all accounts, arising from issues in the design and the implementation of the OF protocol.

1 Introduction

Portable devices such as smart phones and tablets often come with a feature that allows their owner to find those devices when they are lost, typically through the use of a web portal provided by their vendors, such as Google's Find My Device [15], Samsung's Find My Mobile (FMM) [28] and Apple's Find My [2]. A typical requirement for such a feature to work is that the lost device must be connected to the internet so that it can send its location report to a vendor server in the event that its owner flags the device as lost. In recent years, mobile device manufacturers such as Samsung and Apple have extended their lost-device tracking systems with an *offline finding* (OF) feature, which allows a lost mobile device to be found even when it does not have an internet connection. Both Apple and Samsung OF features share two key elements: the use of Bluetooth Low Energy (BLE) for short range transmission of data between devices of a vendor, and crucially, an extensive network of (internet-connected) mobile devices (which we call *helper devices*) that relay location information to a vendor controlled server. We refer to the latter as the *OF network*. The basic idea is quite simple: when a lost device loses its internet connection, it starts broadcasting a unique beacon over BLE, which is then picked up by nearby helper devices participating in the OF network, who then forward the beacon and the location it is found to a vendor server. In this work, we are mainly concerned with Samsung's FMM Offline Finding (OF) feature [29], which was introduced in 2020. An owner can track their devices' locations through Samsung FMM application running in a Samsung mobile device (e.g., a phone or a tablet).

In 2021, Samsung released the Galaxy SmartTag [35], which is a small BLE tracker that can be attached to various items to keep track of their locations. SmartTags are not capable of connecting to the Internet, so they rely on the OF network for long range location tracking (outside the range of BLE). SmartTags are registered and controlled through SmartThings, which is an umbrella control and management platform for a large variety of smart devices and home appliances. OF is also supported for SmartTags using the "SmartThings Find" add-on which works in conjunction with FMM.

Devices in Samsung's OF network can be categorized into three roles: the owner device, the helper device, and the lost device. A mobile device can be registered to the Samsung OF network through the FMM app, while a SmartTag can be registered through Samsung SmartThings app. Each registered device is linked to the owner's account under which it was registered from. When a registered device loses internet connectivity, or in the case of SmartTags, when it is out of the BLE range from its owner device, it broadcasts certain data over BLE periodically. This data contains a rotating identifier, called the *privacy ID*, which is unique to the lost device and which, in theory, can only be linked to its owner by Samsung and the owner device. The helper devices consist of both Samsung devices (phones and tablets), and some third-party devices that support Samsung OF protocol. An active helper device periodically scans for BLE advertisements from nearby

OF devices and reports their locations to a location server. The location reports of the lost devices will be downloaded onto the owner device when the owner queries their locations. The effectiveness of the OF feature depends on the size of its OF network, which in the case of Samsung OF, is estimated to have around 200 million active helper devices [18] in 2022.

In the following, we shall refer to end-user devices, such as mobile phones and SmartTags, that participate in Samsung OF network as OF devices. Our work was driven by the following research questions:

(RQ1) *Identification of an OF device.* Can an OF device be identified through its BLE data?

(RQ2) *Unwanted tracking.* Can Samsung OF network be misused for unwanted tracking of a person or an object by a party other than Samsung?

(RQ3) *End-to-end location privacy.* To what extent does the design of the OF protocol protect the location privacy of the lost devices and the helper devices from the service provider (Samsung)?

(RQ4) *Location report integrity.* Is it possible for an actor (other than the owner and the vendor) to forge a location report of a lost device?

RQ1 concerns the privacy protection of the owner of an OF device against (long term or short term) location tracking through the BLE data emitted by the device by an adversary. RQ2 addresses an attack scenario where a tag is used by its owner to stalk a person without their consent [12]. RQ3 raises the question as to what extent Samsung is aware of the movement of both the owner devices and the helper devices. RQ4 is more of a security (integrity) issue rather than a privacy one and addresses a scenario where the attacker intentionally disrupts the location tracking capability of the OF network through false location reports.

Our main contributions are as follows:

- We provide the first comprehensive reverse-engineering of Samsung OF protocol that allows us to answer definitively the research questions (RQ1 to RQ4) raised above.
- We identified several vulnerabilities that would allow an attacker to link BLE packets observed from a target device over multiple observations, through BLE interactions only, allowing a long term identification of an OF device through its BLE data (RQ1).
- Through our analysis of the OF protocol, we managed to impersonate completely a SmartTag to the OF network. This opens the possibility of creating a custom tracking device that can be tuned to circumvent potential anti-tracking mechanisms by the vendor.
- Our analysis also confirmed that unwanted tracking (RQ2) is possible.
- Our analysis suggests that the vendor does indeed possess the information needed to link an account to a location report (RQ3). Moreover, the vendor server does not appear to check the integrity of the location reports, opening it to manipulation by third parties (RQ4).

Coordinated disclosure. We have reported all the issues we found (see §4) to Samsung and followed an industry-standard of a minimum of 90-day embargo period, prior to the publication of these issues. One of the issues we raised concerns the small pool of privacy IDs being used for FMM BLE packets, which has now been assigned a CVE (CVE-2022-33707, SVE-2022-0126). The more critical security issues that affect the server-side of Samsung OF network, related to SmartTag registration and the location reporting, have been fixed. Section 4.5 provides an update on the status of these issues. Not all the issues we reported have been patched, and in one case, concerning the issue of de-anonymisation through BLE DFU mode (see Section 4.1), the vendor confirmed that they have no plans to issue a fix. However, we believe there is significant public interest in the disclosure of these issues, both from the end user perspective, so that they can make informed decisions on the use of affected devices, and from a scientific perspective, as some of the issues we discovered arise from design choices not specific to a particular vendor.

Related work. The closest to our work is the security and privacy analysis of Apple's FindMy network [21]. Their study uncovered design and implementation flaws that could lead to location correlation attacks and unauthorized access to location histories. They reverse-engineered FindMy protocols and showed that one could create custom tracking devices leveraging on the FindMy network [30].

Several vulnerabilities [11] have been identified in an earlier version of Samsung FMM app, that allow, among others, a malicious app installed in the device to manipulate the URL endpoint accessed by the FMM app and to access unprotected broadcast receivers in the FMM app. This analysis was done prior to the introduction of the offline finding features to FMM, so it did not cover the OF related vulnerabilities.

Both Apple AirTags and Samsung SmartTags use the nRF52 series System on Chips (SoC), which are known to be vulnerable to a power glitching attack. The firmware of both AirTags and SmartTags have been extracted by researchers using this attack [5,27]. While AirTag firmware has been studied extensively [27], we are not aware of any reverse-engineering on the SmartTag firmware prior to our work.

There have been a variety of BLE trackers prior to the introduction of AirTags and SmartTags, such as the Tile tracker. Weller et. al. [34] provides a detailed security and privacy analysis of the security and privacy aspects of these trackers, focusing on their interactions with their associated mobile apps and the backend cloud servers for crowd-sourced location tracking. However, they did not analyze the privacy issues arising from the BLE protocols used in these trackers.

Apple's FindMy network consists of hundreds of millions of active devices, which has raised privacy concerns on whether the network can be abused for malicious tracking. Apple has implemented an anti-tracking framework to detect unknown FindMy trackers for iOS devices. However, Mayberry et al. [25] discovered that Apple's tracking detection mechanism can be defeated through a number of techniques. One technique exploits a "blindspot" in Apple's anti-tracking algorithm that ignores lost mobile devices, focusing only on detecting lost AirTags. As we shall see later, Samsung's anti-tracking feature suffers from a similar oversight.

AirGuard is an anti-tracking application developed by SEEMOO lab to protect Android users from BLE trackers that leverage Apple's FindMy network [20]. It has a higher success rate and lower false positive rate in detecting and reporting trackers compared to Apple's built-in anti-tracking.

A BLE device can be identified by its MAC address. To avoid long-term tracking of a BLE device, vendors often implement anti-tracking mechanisms, such as by randomizing its MAC address. However, issues have been found in implementations of the communication protocols of BLE devices, which can be used to defeat such anti-tracking mechanisms [8–10, 24]. Our own investigations show that many of these issues also affect SmartTags.

Another closely related research area is BLE-based contact-tracing applications, such as Google-Apple Exposure Notification (GAEN) framework [16], DP3T [33], Singapore's TraceTogether and Australia's COVIDSafe [32]. The BLE-related attack surface of these applications, i.e., in relation to RQ1, is similar to our findings, e.g., attacks discussed in [3, 7, 22, 32] show that devices running these applications can be linked through their BLE data. Another similarity is the "wormhole" relay attack demonstrated in [3], which is similar to the relay attack discussed in Section 4.2. There is, however, a crucial difference between an offline location tracking system and a contact-tracing system. In the latter, the "location" information (which takes the form of a list of anonymized user IDs detected in proximity, without any geolocation information) is not sent to users of the applications, but only to a designated health authority. This essentially makes the RQ2, i.e., the possibility of unwanted tracking irrelevant.

Outline. The remainder of the paper is structured as follows: Section 2 gives a brief overview of relevant cryptographic and BLE related concepts. Section 3 presents technical details of the OF protocol that result from our investigations. This section covers the OF operations for SmartTags. The OF operations for FMM devices are a much simplified version of the SmartTag protocol, so its security and privacy issues are subsumed to those of SmartTags. The interested reader can consult Appendix B for details of the FMM protocol. In Section 4, we perform a security and privacy analysis on Samsung OF protocol based on our findings discussed in previous sections. In Section 5 we discuss the broader implication of our findings on the design of offline finding protocols in general. Section 6 concludes the paper. For clarity, our presentation of the OF protocol abstracts away some concrete details such the data format and the BLE interface used, but these details are available in the appendices.

2 Background and methodology

This section gives a very brief overview of the relevant cryptographic functions used in the Samsung OF protocols and some basic concepts related to BLE.

ECDH key exchange and AES block cipher. There are two main cryptographic constructions used in the OF protocol: the Elliptic-curve Diffie-Hellman (ECDH) key exchange protocol and the AES block cipher and its associated encryption modes. We explain briefly each of these constructions. For further details, we refer interested readers to [19] for ECDH and [13] for the AES algorithm.

The ECDH builds on the Diffie-Hellman key exchange protocol [14], where the underlying group operations are defined over an elliptic curve (EC). Samsung's OF implementation of ECDH uses the elliptic curve Curve25519 [4], which was designed to achieve high speeds at computation without compromising the security strength. The Advanced Encryption Standard (AES) algorithm [13] is a symmetric block cipher that is widely used for data encryption, and as a building block for other cryptographic functions. Samsung's FMM and SmartTags implement AES CBC mode cipher with PKCS#7 padding scheme [26] to encrypt/decrypt data for various OF related operations.

Bluetooth Low Energy (BLE) SmartTags uses BLE [17], which is a short-range wireless communication technology, for data transmission. The protocol stack of BLE consists of various layers and profiles, of which, the most relevant ones to this work are the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT). GAP defines the procedures for device discovery and connection establishment. A BLE device can operate in one or more of the following roles:

- Advertiser: a device that sends out BLE data that is available to any nearby Bluetooth capable devices.
- Observer: a device that listens to BLE advertisement data and may process the data from advertisers.
- Central: a device that initiates a connection after receiving advertisement data from an advertiser.
- Peripheral: a device that accepts the incoming connection from a central.

GATT defines the data organization and exchange over connections between BLE devices. GATT uses a hierarchical structure to organize data. A GATT profile may contain multiple services, each contains one or more characteristics. Each characteristic is a container of user data. A characteristic can

be followed by descriptors, which provide additional metadata of the characteristic and its value.

BLE has two ways of transferring data: advertising over BLE and data exchange over connections. Advertising is the process of a BLE device sending out data packets in one-way, while communication over connections allows bidirectional data transfer between the peripheral and the central. Data packets are exchanged through characteristics in the GATT server of the peripheral device. A BLE device is addressed through its MAC address, which is a 48-bit identifier. There are four types of MAC addresses: Public Address, Random Static Address, Random Private Non-Resolvable Address, and Random Private Resolvable Address. A Public Address is registered with IEEE and never changes. A Random Static Address is not registered and remains constant during device runtime. Each Bluetooth-capable device has an *Identity Address*, which is either a Public Address or Random Static Address. The two types of Random Private Addresses (Non-Resolvable, Resolvable) are used for privacy protection purposes. Random Private Non-Resolvable Addresses are generated completely randomly, whereas Random Private Resolvable Addresses (RPAs) are generated using a key-hashed function from a random seed value and a 16-byte key called the Identity Resolving Key (IRK). The possession of the IRK of a device would also allow one to de-anonymize its RPAs.

Pairing is the process by which two BLE devices exchange necessary information so that an encrypted connection can be established. BLE has several pairing modes, which are determined by the authentication requirements and input/output (IO) capabilities of the pairing devices. As part of pairing, the IRKs are exchanged, so the devices can identify their respectives RPAs using the IRKs. BLE supports different pairing methods to authenticate participants in the pairing procedure. The simplest pairing method, called *Just Works*, does not check the authenticitiy of the participants, and has been exploited to steal the IRK of a target device stealthily [32,36,37].

Methodology. A combination of investigation approaches were used to understand the OF protocol for SmartTags and the FMM app. Devices involved in the analysis include a research laptop equipped with a BLE 4.2 adapter running Ubuntu 20.04 LTS, a number of Samsung mobile phones running Android versions 8.0 - 12, and SmartTags with firmware version 1.01.26 and 1.02.06. We used a combination of application and firmware reverse engineering to study the innerworking of various protocols involved in the OF network, analysis of various logs produced by Android systems and applications, and analysis of both BLE and network traffic between devices and vendor servers. The methodology is quite standard for vulnerability research so we omit the details here.

3 The offline finding protocol

This section discusses the key findings on the Samsung offline finding (OF) protocol. Here we discuss only the OF protocol

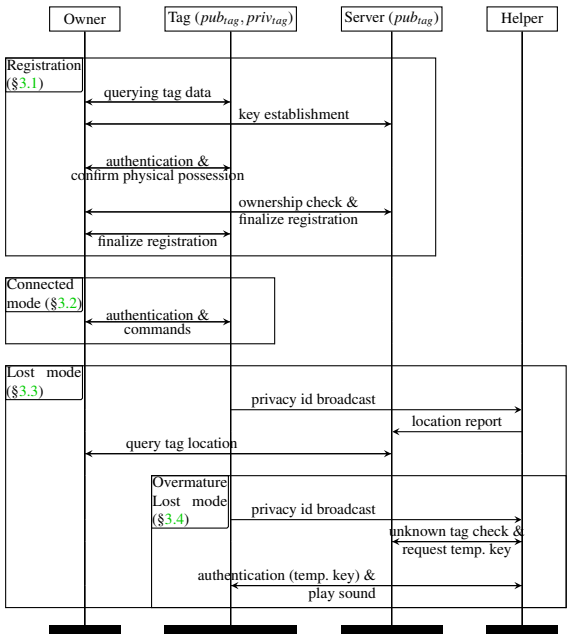

Figure 1: An overview of Samsung OF protocol for SmartTags

for Galaxy SmartTags, but details of the OF protocol for FMM, which is a simplified version of the SmartTag protocol, are available in Appendix B. Our analysis was performed on SmartTags with firmware versions 1.01.26 and 1.02.06, and the FMM app versions prior to version 7.2.24.12.

The OF protocol for SmartTags involves both online interactions (over the Internet) with various vendor servers and offline interactions (over BLE) with nearby tags and mobile devices. We discuss here four important subprotocols, which are summarised in Figure 1. The communication between the devices and the server is done through HTTPS, which we assume to be secure. There are four principals involved: the owner device, the tag, the vendor server and the helper device. The subprotocols are as follows:

1. The registration protocol (§3.1). This protocol involves the owner device, the tag and the server. The owner initiates the protocol by acquiring relevant tag data, such as the serial number, firmware version, etc., and initiates a key establishment protocol with the server, to derive various symmetric keys that will be used in subsequent interactions with the tag.

2. The protocol for tags in the *connected mode* (§3.2). This protocol is executed right after the registration, and whenever the tag is in the proximity of the owner device after having been out of the BLE range. It contains subprotocols for authenticating the owner device to the tag and vice versa. Once authenticated, various commands and data may be exchanged over BLE.

3. The protocol for tags in the *lost mode* (§3.3). This protocol is triggered after the tag has lost its BLE connection to its owner for less than 24 hours. In this state, it broadcasts anonymized rotating *privacy IDs* that trigger nearby helper devices (who are scanning for privacy IDs periodically) to record and report the locations of the recorded privacy IDs to the server.

4. The protocol for tags in the *overmature lost mode* (§3.4). A tag transitions to the *overmature lost mode* after being lost for over 24 hour. A helper device that detected a tag in the overmature lost mode would initiate an anti-stalking detection process. This process aims to enable the helper device to locate a (potentially) tracking tag by playing sound on the tag. To play sound, the helper would need to obtain a temporary key from the server to authenticate itself to the tag.

3.1 SmartTag registration

The SmartTag registration protocol requires interacting with an unregistered tag and the vendor server. For the latter, there are actually multiple servers involved, providing services such as user authentication, application related services such as remote attestation, and services related to storing and retrieving location reports. For simplicity, we shall refer to these servers collectively as the vendor server (or simply "the server") in the following discussion.

The interaction between the owner device and the tag is done through BLE using BLE advertisement and a GATT profile. A SmartTag uses two UUIDs to advertise its presence over BLE: FD59 for non-registered tags, and FD5A for registered tags. SmartTags do not support internet connectivity and thus rely on the owner device (typically a mobile phone) to perform various setups over BLE connections. This is done through its GATT profile, which defines various services and characteristics that the tag and its owner device use to exchange data and commands. In the following, we shall omit the concrete UUID used for each characteristic in the GATT profile, and refer to it using a symbolic name instead. But detailed characteristic UUIDs can be found in Appendix C on arXiv. The SmartTag GATT profile has four primary services which can be summarized as follows:

Authentication Service The Authentication Service uses three characteristics, `NONCE`, `ENONCE` and `SUPPORTED_CIPHER`, for authenticating a connected device over BLE.

DFU Service Service UUID FE59 is a part of the nRF52833 Buttonless Secure DFU service for over-the-air firmware updates. It has a writeable characteristic (`BUTTONLESS_DFU`) that can be used to reboot the tag.

Onboarding Service Service UUID FD59 is used for device onboarding/registration activities. During the registration process, the owner device and the tag exchanges configuration and cryptographic data over various characteristics under this service.

Command Service Service UUID FD5A is primarily used for performing more complicated interactions between the owner device and a tag, such as executing a supported command (e.g., alarm, changing ringtone) on the tag.

A SmartTag registration is initiated by an owner device running the SmartThings app. It involves online interactions with the server, and offline interactions with the tag. The interaction with the server requires a valid user account. In the following discussion, we shall assume that a valid user account has been created and an authenticated session between the owner device and the server has been established. The registration protocol consists of five stages: key establishment, owner-tag authentication, confirmation of physical possession of the tag, tag ownership check, and registration finalisation.

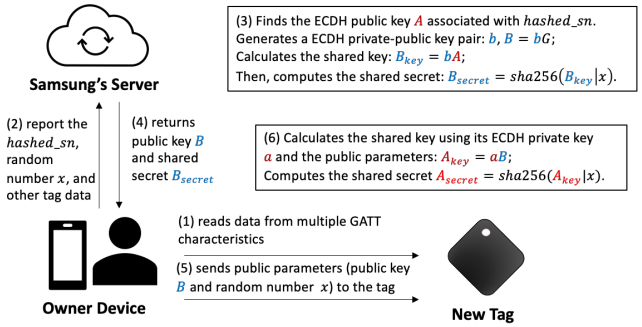

Figure 2: Shared secret establishment protocol

Stage 1: key establishment The first stage of the registration protocol is essentially a key establishment protocol between the tag and the server, mediated by the owner device. Note that since we do not have access to the server code, our analysis of this stage is based on our analysis of the SmartTag firmware and the intercepted traffic between the owner device and the server. Our findings suggest that the tag keeps a pair of private-public ECDH key (a, A_{pub}), which is fixed for the lifetime of the tag and that the server keeps at least the public key of each tag. A tag is identified uniquely via its serial number, which is the identity address of its BLE controller. The public-private key pair of the tag is never sent out from the tag or the server. The shared secret establishment protocol is summarized in Figure 2, which we elaborate below.

Step 1. The owner device obtains the necessary registration data from the tag, through the tag's advertisement payload and Onboarding Service. Among this registration data is a hashed serial number (`hashed_sn`) unique to the tag, which corresponds to the SHA256 digest of the BLE identity address of the tag.

Step 2. The owner device generates a 32-byte random value x, and sends x along with the registration data obtained from Step 1 to the server.

Step 3 & 4. After receiving the request, the server looks up the public key of the tag A_{pub} associated with the `hashed_sn`. The server then generates an ephemeral private-public key pair (b, B_{pub}), and computes the ECDH shared key $B_{key} = bA$. Finally, the shared key is concatenated with the random number x to form the input for the SHA-256 hash function to produce the shared secret: $B_{secret} = SHA256(B_{key}|x)$. The server then sends B_{pub} and B_{secret} to the owner device.

Step 5. The owner receives B_{pub} and B_{secret} from the server, and forwards B_{pub} and x, to the tag.

Step 6. The tag receives B_{pub} and x from the owner device and computes $A_{key} = aB$ and $A_{secret} = SHA256(A_{key}|x)$.

By the property of ECDH, assuming no tampering from an adversary, at the end of Step 6, we should have $A_{secret} = B_{secret}$. That is, all participants now share the same secret B_{secret}. This shared secret will be used next to compute several AES keys that will be used in subsequent protocols.

The first 16 bytes of the shared secret B_{secret} are taken as the `masterSecret`. It is used to derive six 16-byte subkeys for securing communication between the owner device and the tag. Note that Samsung OF protocol does not use any default BLE pairing and authentication mechanisms, so this shared secret is unrelated to BLE Long Term Key (LTK) that is normally exchanged as part of BLE pairing protocols [17].

The subkeys are derived using a key derivation function, given below, that combines the `masterSecret` and a parameter that is used to differentiate subkeys:

$$kdf(k,x) = SHA256(m(k,x))[0:15]$$

where $m(k,x) =$

Bytes 0-15	Bytes 16-19	Bytes 20-
k	00000001	x

.

The following four subkeys are computed by the owner device, the tag and (presumably) the server.

1. **Owner authentication key**: This key is used by the owner to establish an authenticated BLE session with a SmartTag. It is computed by applying :

$$AK_o = kdf(\text{masterSecret}, \text{"bleAuthentication"})$$

2. **Owner GATT key**: This key is used for encrypting the data exchanged in the GATT interactions between the owner and the tag. It is dependent on a nonce and is valid for a single session of interactions with the tag.

$$GK_o(nonce) = kdf(\text{masterSecret}, nonce)$$

Here *nonce* is a 16-byte value received from the SmartTag during each BLE authentication process, see §3.1.

3. **Privacy key** : This key is used for generating unique privacy IDs for a SmartTag (see §3.3).

$$PIDK = kdf(\text{masterSecret}, \text{"privacy"}).$$

4. **Advertisement signing key** : This key is used for signing and validating the integrity of the BLE data broadcasted by a SmartTag:

$$ASK = kdf(\text{masterSecret}, \text{"signing"})$$

Two additional subkeys are derived by a SmartTag and the server, but not the owner device (see §3.4 for details of when and how these keys are used).

- **Non-owner authentication key**. This key is used by a non-owner device to authenticate to the tag.

$$AK_{no} = kdf(\text{masterSecret}, \text{"nonOwner"})$$

- **Non-owner GATT key**. This key is used in a GATT session between a non-owner and the tag for exchanging commands. It is dependent on a nonce that is exchanged during the GATT interaction.

$$GK_{no}(nonce) = kdf(AK_{no}, nonce).$$

Notice that unlike the owner GATT key, the non-owner GATT key is not generated directly from the masterSecret; rather it is derived from the non-owner authentication key.

Stage 2: Owner-tag authentication After computing the `masterSecret` and the AES keys, the owner device initiates a two-way authentication with the tag to establish an authenticated connected session. This protocol is implemented through BLE interaction only, using the Authentication Service of the GATT profile of the tag. In the protocol description below, O denotes the owner device and T denotes the tag. Throughout the remainder of the paper, we shall use the notation $E_k(x,y)$ to denote the AES/CBC/PKCS7 encryption of plaintext y with key k and initialization vector x.

1. $O \rightarrow T$: n_O
2. $T \rightarrow O$: n_T
3. $O \rightarrow T$: $E_{AK_o}(n_T, \text{"smartthings"})$
4. $T \rightarrow O$: $E_{AK_o}(n_O, \text{"smartthings"})$

Here n_O and n_T refer to nonces generated by O and T, respectively. At Step 3, the tag checks that the received ciphertext is indeed the encryption of the text "smartthings"; likewise in Step 4, the owner checks that the received ciphertext is of the expected form. If any of these checks fail, the authentication fails; otherwise the authentication is established, and both the tag and the owner derive an Owner GATT key using n_T, i.e., $gk_o = GK_o(n_T)$. This key acts as a session key that is used to secure data transmission in this authenticated session, until the BLE connection is terminated.

Stage 3: confirming physical possession of the tag. After the owner and the tag have successfully established an authenticated BLE connection and derived the session key gk_o, the next step is to establish the physical presence of the tag. In a normal registration flow, the SmartThings app will ask the user to press the tag button to ensure physical possession of the tag. Pressing the tag button sets the value of the `CONFIRM_STATUS` characteristic on the tag to 0x01 (encrypted using gk_o with IV set to n_T) from the default value 0x00.

$$T \rightarrow O \ : \ E_{gk_o}(n_T, 0x01)$$

The owner's device would only continue the registration flow after validating the value of this characteristic.

Stage 4: tag ownership status check This stage checks the ownership status of the tag to ensure that it is not currently registered to another user. This is done via a simple request to the server, containing the serial number *sn* of the tag. If the server indicates that the tag has already been registered to another account, the registration process will abort.

Stage 5: finalizing tag registration This stage creates an online profile of the tag, associated with the owner's account. The protocol can be described abstractly as follows (a more detailed version can be found in Appendix C on arXiv), where S denotes the server.

1. $O \rightarrow S$: sn, id, B_{secret}
2. $S \rightarrow O$: $deviceId, metadata$
3. $O \rightarrow T$: $E_{gk_o}(n_T, metadata)$
4. $O \rightarrow T$: $E_{gk_o}(n_T, curtime)$
5. $O \rightarrow T$: $E_{gk_o}(n_T, \text{"Finish"})$
6. $T \rightarrow O$: pid
7. $O \rightarrow S$: pid, loc

In Step 1, the owner device sends a record, containing, among others, the tag serial number *sn*, an identifier *id* and the shared secret B_{secret} established in §3.1. Recall that *sn* is used in the ownership status check in the previous stage, to avoid a tag being registered twice. Our observations showed that *sn* and *id* were identical and equal to the tag BLE identity address.

In Step 2, the server returns a unique *deviceId* that is linked to the tag, and a *metadata* record that contains the following fields: the *privacy Id pool size* (*pidsize*), the *privacy Id seed* (*pidseed*) and the *privacy Id IV* (*pidIV*). The owner device keeps a record of these parameters and in Step 3, forwards them (encrypted) to the tag. These parameters are used later by the tag to generate BLE advertisements. The *deviceId* can be used later to lookup the location of the tag.

In Step 4, the owner device sends the current time information (in UTC) to the tag so the tag can synchronize its time with the owner. Finally, the owner sends an encrypted message (`"Finish"`) to indicate the end of the registration process and disconnect from the tag in Step 5.

In Step 6, the tag broadcasts BLE data containing a privacy ID (see §3.3) that the owner device uses to identify it. After discovering its own tag, in Step 7, the owner device reports the current location *loc* and the tag *pid* to the location server. The first location report of a tag made by its owner device creates a device profile on the server, which bonds the tag's ownership status with the owner's Samsung account, preventing others from registering the tag (§3.1). Crucially, the server uses the value *id* as the identity of the tag for the ownership bonding.

3.2 Owner-Tag interaction

A tag may move in and out of the BLE range of its owner device. The owner device (which also acts as a helper device for tags it does not own) periodically scans for tags in its proximity. When a tag is detected in the proximity of its owner, after having been away, the owner (through the SmartThings app) will automatically initiate the owner-tag authentication protocol over BLE, as described in §3.1. Upon successful authentication, the app will show the tag as connected and the owner can perform various supported commands on the tag, such as ringing the tag. SmartThings also allow an option to configure the tag to perform limited actions on the owner device, such as ringing the owner device when a physical button in the tag is pressed. Appendix C on arXiv contains details of some of these commands. Each command is triggered by exchanging encrypted data $E_{gk_o}(n_T, Data)$, where gk_o is the owner GATT key and n_T is the tag's nonce sent during the owner-tag authentication stage (§3.1), and *Data* is an encoding of the command:

$Data =$

Bytes 0-3	4	5-
Counter	Opcode	Argument(s)

The counter value corresponds to the total number of commands successfully sent by O during the current authenticated session. The opcode specifies the command type, as a characteristic may support multiple commands, e.g., opcode 0 for alarm off, 1 for alarm on.

3.3 Location querying and reporting

We now discuss the subprotocols for an owner device to query its lost tags from the location server, and for a helper device to report location information of lost tags to the server. The location querying protocol consists only of simple POST requests to an URL under the `api.samsung.com` domain, so we shall omit the details. In the following, we focus on subprotocols used for location reporting.

SmartTag BLE advertisement data. The helper device identifies nearby (lost) SmartTags through their BLE advertisement data only, so we first present details on the advertisement structure and how it is generated. Figure 3 shows the

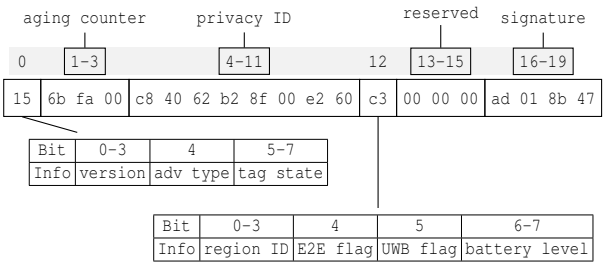

Figure 3: The OF advertisement structure for SmartTags

Bits 5-7	Name
001 (1)	Premature lost mode
010 (2)	Lost mode
011 (3)	Overmature lost mode
100 (4)	Paired with one device
101 (5)	Connected to one device
110 (6)	Connected to two devices

Table 1: Operating states of a SmartTag

OF advertisement structure for SmartTags with the following fields highly relevant to the OF protocol: the *tag state* (byte 0), the *aging counter* (bytes 1-3), the *privacy ID* (bytes 4-11), and the *signature* (bytes 16-19).

The tag state is encoded in bits 5-7 of byte 0 and denotes the operating states of a tag. There are six different tag states, shown in Table 1. The state of a registered tag becomes premature lost once it is disconnected from its owner device or rebooted. After operating in premature lost mode for 15 minutes, the state changes to lost, which informs nearby helper devices that it is considered lost. After 24 hours in lost mode, the state becomes overmature lost, where certain BLE operations slow down for power saving. A helper device that finds a tag in the overmature lost mode will initiate an anti-tracking process (see §3.4).

Bytes 1-3 store the aging counter, a timestamp computed using the tag's time (UTC) and a hardcoded constant: $agingCounter = (tagTime - 1593648000)/900$. The aging counter is universal for all SmartTags in-sync with the server's time and updates periodically.

Bytes 4-11 store an 8-byte privacy ID, which is a unique identifier of a SmartTag. Each registered SmartTag has a set of unique privacy IDs called the privacy ID pool. This set is deterministically generated using the privacy key (*PIDK*) and the privacy ID configurations (*pidsize*, *pidseed*, *pidIV*) that the server sent to the owner during the finalization stage in the registration process (§3.1). The privacy IDs are an essential part of the OF protocol as they associate a tag with its owner's Samsung account. For each $i \in \{1, \ldots, pidsize\}$, the privacy ID pid_i is generated as follows:

$pid_i = E_{PIDK}(pidIV, input_i)$, where $input_i$ is the byte array

Byte 0	Byte 1	Byte 2-9	Byte 10	Byte 11
x_i	y_i	*pidseed*	x_i	y_i

$x_i = (i \gg 8) \wedge 256$ and $y_i = i \wedge 256$. Here, $\gg$ represents right bitshift and $\wedge$ denotes bitwise AND.

The privacy pool size for SmartTags is 1000 when we made our analysis, but the value is controlled by the server so it is not firmware-specific.

The signature field at the end of the advertisement data serves as a cryptographic checksum for the first 16 bytes. Let *blePayload* denote the first 16 bytes of the BLE advertisement data. Then the four signature bytes are obtained from the first 4 bytes of the $fullSignature$ defined below:

$$fullSignature = E_{ASK}(pidIV, blePayload)$$

where *ASK* is the advertisement signing key (see subkey 4). Any changes to the first 16 bytes of the advertisement will likely cause the signature bytes to change correspondingly, which allows the integrity of the BLE data to be validated by parties with the privacy ID configuration and the shared secret of the tag, such as the owner and the server.

A tag rotates its advertisement data periodically. A tag in any non-overmature lost mode updates its privacy ID, aging counter (incremented by 1), and signature every 15 minutes. Under the overmature lost mode, a tag updates the aging counter and signature every 15 minutes. However, the frequency for shuffling the privacy ID reduces from once every 15 minutes to once every 24 hours.

Location reporting A helper device regularly scans for BLE advertisement data from nearby SmartTags. It filters BLE advertisements based on the advertising UUID for SmartTags (FD5A). The helper device stores the privacy IDs of lost devices in a local database that can store up to 1000 entries. Note that these privacy IDs do not necessarily represent distinct devices, as a tag can generate multiple privacy IDs and the (honest) helper device does not have the privacy key (*PIDK*) needed to link these IDs. A privacy ID of a tag is marked as expired if it has not appeared in the BLE scanning for 15 minutes and will be removed from the database. Moreover, the helper device only reports locations of SmartTags in lost or overmature lost mode.

The helper device will report geolocations of lost SmartTags in the database based on estimated locations received from the WiFi or GPS service. Through reverse engineering and runtime analysis on a helper device, we found that each helper device has a pair of RSA 2048-bit private signing key (pk_H) and its corresponding verification key (pub_H) stored in the device, secured using Android keystore. Our analysis shows that these keys are embedded in a native library contained in the FMM apk, so they are likely not unique per device. The public key pub_H is signed by an intermediate CA owned by Samsung. These keys are used in the location report protocol below to certify the originality of the report

(in that it originated from an official Samsung device rather than an unauthorized third party).

To submit a location report, the helper must first obtain an access token from the server. This token request process is re-used again in another protocol (for interacting with an unknown tag), so we describe it separately here.

1. $H \rightarrow S$: REQ
2. $S \rightarrow H$: n_S
3. $H \rightarrow S$: $pub_R, n_S, sig(n_S, pk_H), cert(pub_H)$
4. $S \rightarrow H$: $access\_token$

In Step 1, the helper device generates a pair of private-public key (pk_R, pub_R) and then sends a request for location report to the server S. Concretely this step is done via a simple HTTPS GET request to S. The server responds with a 16-byte nonce, encoded as a hexadecimal string. H extracts the signing key pk_H from its keystore, after having performed an attestation protocol, and uses it to sign the nonce n_S. The helper then sends the nonce n_S, its signature $sig(n_S, pk_H)$, the certified public key $cert(pub_H)$, and pub_R to server. Note that the key pair (pk_R, pub_R) is not used in the location report but will be important later in another protocol. If the server checks the validity of the certificate and verifies the signature and sends a unique access token (which is a JWE token, tied to the nonce n_S) (Step 4) if everything checks out. The access token is then used to authenticate the location report.

$$H \rightarrow S \ : \ access\_token, report$$

where $report$ contains a list of advertisement data and the geolocation they were detected. Each location report allows a maximum of 5 recently found devices ($time_{found} \geq time_{current} - 1$ (minute)) from the local database to be reported. Each access token has a fixed expiry time (around 32 hours, based on our experiments), after which, the token request protocol above must be repeated to obtain a new token.

3.4 Unknown tag detection

The SmartThings app has a feature for detecting nearby overmature lost mode tags for anti-stalking purposes. If such as tag has been detected to be following a helper device, the SmartThings app gives the device owner an option to play sound on the tag to help locating it. Unlike AirTags where any user can issue a command to the tag to play sound [21], a SmartTag only responds to commands from an authenticated device. Since the helper device does not have the authentication key to authenticate to the tag, it would need the vendor server's help.

We assume that the helper device has performed the protocol to request JWE access token (see the location reporting protocol above) and obtained the token. To play sound on the tag, the helper needs to initiate a GATT connection and reads off a nonce from a certain characteristic. In the following, the notation $AE_{pub}(x)$ denotes an asymmetric encryption of plaintext x with public key pub.

1. $H \rightarrow T$: n_H
2. $T \rightarrow H$: n_T
3. $H \rightarrow S$: $access\_token, n_T, pid$
4. $S \rightarrow H$: X, Y
5. $H \rightarrow T$: en_T
6. $T \rightarrow H$: $E_{AK_{no}}(n_H)$
7. $H \rightarrow T$: $E_{gk_{no}}(n_T, command)$

In Step 4, the messages are computed as follows:

$$X = AE_{pub_R}(E_{AK_{no}}(n_T)) \qquad Y = AE_{pub_R}(GK_{no}(n_T))$$

where pub_R is the public key H generated in the access token request protocol. H decrypts these to obtain $en_T = E_{AK_{no}}(n_T)$ and $gk_{no} = GK_{no}(n_T)$ used in the remainder of the protocol. If we omit step 3 and 4, this protocol is similar to the owner-tag authentication protocol, except that the authentication key used is AK_{no}, and the GATT key used to send the command is $GK_{no}(n_T)$. Note that the server never discloses the AK_{no} itself. This request needs to be accompanied by a current access token, and an identifying information from the tag (i.e., its privacy ID pid, and some other information we omit here).

4 Security and privacy analysis

The four research questions guide the construction of our adversary model (Table 2) which we use for our analysis. This model classifies potential threats to the OF system into four categories based on our research objectives, extending the FindMy adversary model proposed in [21]. Firstly, our model subdivides proximity-based threats (A1) into passive (A1.1) and active (A1.2) categories, based on the level of interaction with the SmartTag's BLE interfaces. In line with the FindMy model, we consider network-based (A2) and service operator (A3) threats. Finally, we introduce a new Tag Owner category (A4), acknowledging potential security implications posed by SmartTag owners themselves. The final column indicates the related research questions, highlighting the implications of each threat category.

We now analyze the security and privacy issues affecting Samsung's OF system in alignment with our research questions, each addressed in the following sections: §4.1 (RQ1), §4.2 (RQ4), §4.3 (RQ3), and §4.4 (RQ2). Then, we provide an update on various bugs discussed in Section 4.5.

4.1 Proximity-Based Attack (A1-RQ1)

Attack Scenario By passively eavesdropping on BLE advertisements (A1.1) or actively engaging with the SmartTag's GATT server (A1.2), the attacker could identify and track the presence of the neighbour's FMM device or SmartTag, thereby gaining insights into their daily schedule.

Table 2: Adversary Model for Samsung SmartTags

Model	Assumptions	Capabilities	Impact
Passive Proximity-based (A1.1)	Within BLE communication distance with a tag; Controls a Bluetooth device	Record and replay BLE advertisements	RQ1 (§4.1)
Active Proximity-based (A1.2)		Interact with tag's GATT server	
Network-based (A2)	MITM position between Samsung server and a tag.	Intercept, redirect, or modify network traffic	RQ4 (§4.2)
Service Operator (A3)	Access to backend systems.	Access to all location reports and secret keys for each registered SmartTag.	RQ3 (§4.3)
Tag Owner (A4)	Owns a SmartTag; Controls a Bluetooth device; Direct contact with a victim	Hide the tag/customized tracking device in victim's belongings	RQ2 (§4.4)

Linkability Flaws The privacy ID pool size for OF devices is specified through a parameter sent by the server during the registration phase. For FMM devices, this is 51, while for SmartTags, it is 1000. As the pool size is not hardcoded on the client side, this could change in future updates to the server. As it stands currently, even with 1000 pool size, there is a high probability (due to the birthday theorem) that the same privacy ID could be reused in approximately $\sqrt{n}$ rotations, where n is the pool size. From our experiments, a few days of passive observations of the BLE data would cover all 51 privacy IDs of an FMM device. Recent versions of the FMM app have added a layer of obfuscation to the privacy IDs, keeping the same privacy ID pool size. However, our preliminary finding shows this can be easily de-obfuscated (see Appendix B.2.3). This obfuscation is not implemented in SmartTags.

GATT server leaking sensitive data. A registered SmartTag advertises on RPAs that frequently change. However, we have found that sensitive information leaked by characteristics in SmartTags' GATT profile (Appendix C.1 on arXiv) provides two ways for an attacker to de-anonymize the tag's identity: The `IDENTIFIER` characteristic contains its identity MAC address, and the `HASHED_SERIAL_NUMBER` characteristic contains the SHA256 hash of the identity address. Both values are static and unique for each tag and readable, which can be used by any nearby adversaries to identify the tag.

The `SUPPORTED_CIPHER` characteristic is readable and writable. It contains a default value `"AES128/CBC/PKCS7"` that specifies the cipher being used for BLE authentication. However, we discovered that writing custom values to this characteristic would overwrite the default value until the tag is restarted. Hence, an adversary can identify a registered tag by writing a custom identifier to this characteristic.

DFU device reboot. The Galaxy SmartTag has a DFU (Device Firmware Update) Service for secure over-the-air Firmware updates. We discovered that a SmartTag can be rebooted to the DFU mode by enabling indication on the Buttonless DFU characteristic and writing byte 0x01 to it [31]. This is actually not part of Samsung implementation; rather it is a default service available to nRF52 chipset for firmware updates so this vulnerability is not specific to SmartTags.

In the DFU mode, the tag advertises on a random static MAC address and waits for new firmware packages. If no data is received over a short period (approximately two minutes), the tag will reboot and resume its lost mode BLE operations. Additionally, the DFU mode reveals the identity MAC address of the tag - the MAC address used in the DFU mode ($addr_{DFU}$) equals to the identity MAC address ($addr_{Id}$) plus one [31], e.g., if $addr_{DFU}$ is observed to be 11:22:33:44:55:66, it can be inferred that $addr_{Id}$ is 11:22:33:44:55:65. However, after coming out of the DFU mode, the aging counter in the OF data is reset to 0, making OF tracking unavailable for the tag since the aging counter will be considered as too old and its location will not be accepted by the server.

Unintended pairing with a SmartTag. The update to firmware 1.02.06 introduces a new vulnerability. A SmartTag with this new firmware appears to accept pairing request, using the Just Works association mode, allowing the attacker to obtain the IRK and the identity address of the tag. The IRK can then be used to resolve the RPAs the tag use in BLE advertising. The IRK appears to be persistent across reboot and account switching. So removing the tag from a Samsung account and pairing with another account does not reset the IRK. The possession of the IRK allows a more stealthy tracking of the tag, as the attacker does not need to connect to tag; they simply observe the RPAs used to advertise the payload and de-anonymize them using the IRK.

Unintended Silent Pairing with an Owner Device. It is possible to impersonate a SmartTag and silently pair with its owner device by exploiting the following pairing behavior in the BLE specification: if a central device encounters an "Insufficient Authentication" error when interacting with a characteristic in a peripheral, it will initiate the pairing procedure with the peripheral. This behavior has been exploited in previous work [32, 36, 37] to initiate unintended pairing. As noted in [37], the attacker can influence the association method for the pairing, e.g., force the pairing to use a less secure method, such as Just Works.

In this attack, the attacker acts as the peripheral, while the central device is the owner device. The attacker creates a GATT profile of a SmartTag and replay the latest BLE advertisement from the tag to trick the owner device into initiating the BLE Authentication procedure (§3.1). BLE Authentication starts with the owner device writing to the `NONCE` characteristic of the (impersonated) SmartTag's GATT server. By set-

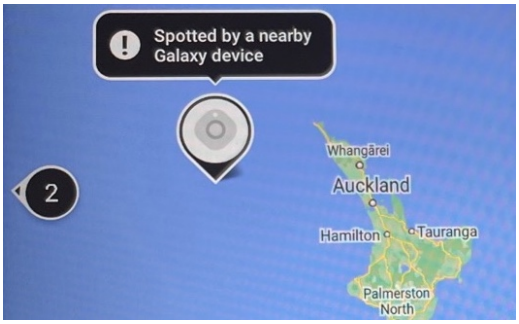

Figure 4: Fake location being updated on the owner device

ting the write permission for the `NONCE` to `encrypted-write`, the "Insufficient Authentication" error will be triggered upon write requests. Prior to the November 2020 patch [1], pairing is performed silently on most Android versions if Just Works is used [32]. Older models of Samsung devices that are not eligible to receive the update, such as Samsung Galaxy 7, remain vulnerable. This attack allows the attacker to obtain the IRK and the identity address of the owner device that can be used for long-term tracking.

4.2 Network-based Attack (A2-RQ4)

Attack Scenario In the context of network-based threats, an attacker stealing a tag (attached to a stolen item) may be able to hide the location of the stolen item by forging location reports using the tag's lost mode advertisement, leading the owner to a false trail.

Forging location reports We found several ways in which the integrity of the location report can be compromised, allowing an attacker to report fake locations of a lost tag.

Relay attack. Recall that the helper device simply forwards the BLE advertisement data of a lost tag or mobile device to the location server. There are no mechanisms for the helper to determine whether the data was indeed broadcasted by a nearby legitimate OF device. This allows a very simple relay attack: two attackers A and B in two different locations can collude by forwarding the BLE advertisement data of a device observed in A's location to B to be replayed at B's location, and vice versa. If there are helper devices in both locations, they will submit conflicting reports. Indeed, our experiments show the location server does not check for the consistency of the location reports, e.g., one device could appear to be detected in two different continents within seconds. This attack is especially effective against FMM devices as their advertisement data has no expiry time and can be replayed indefinitely. In contrast, with SmartTags, the aging counter in the advertisement prevents an indefinite replay. From our observation, we've noticed that advertisement data that is more than 7 days old will be rejected by the server.

Notably, this appears to be a universal issue affecting all crowd-sourced BLE tracking systems, including Apple's FindMy. Potential mitigation will be discussed in §5, such as using a distance-bounding protocol. However, we emphasize that this is a difficult problem that extends beyond the scope of this paper.

The next three attacks exploit the location report protocol itself. They correspond to three different ways in which the attacker can obtain an access token to submit location reports (see §3.3). Recall that the location report will only be accepted by the location server if the reporter has a valid access token. These attacks are more powerful as they do not require any helper devices to be present at all at the target location. For example, we were able to forge a report of an OF device detected in the middle of an ocean, as shown in Figure 4.

Re-using location reporting access token. As it turned out, once the helper device receives an access token from the vendor's server for location reporting, it records the token in its system log. Consequently, an attacker who owns a helper device can extract this token from the log and send fake location reports using it. This process can be easily automated as the report is done through a simple HTTPS POST request to the location server.

Access token renewal through signature replay. We discovered that in the access token request protocol, the server does not check whether the nonce n_S it sent in Step 2 is the same as the n_S it receives in Step 3. This allows an attacker who is in possession of a signed nonce from a previous session to replay it to get another access token. More specifically, suppose the attacker has n'_S and $sig(n'_S, pk_H)$. Then the following is a valid protocol run for token renewal:

1. $H \rightarrow S$: REQ
2. $S \rightarrow H$: n_S
3. $H \rightarrow S$: $pub_R, n'_S, sig(n'_S, pk_H), cert(pub_H)$
4. $S \rightarrow H$: $access\_token$

even when $n_S \neq n'_S$. Note that as all communication between the helper device and the location server are secured via HTTPS, for this attack to be possible, the attacker would need to be able to decrypt the TLS encrypted traffic. In our experiments, this was done by installing a root CA in a rooted helper device, and perform MITM attack between the device and the server. We observed that in recent Samsung phones running Android 10 or later, the location report process involves an attestation protocol before the phone could extract the signing key to sign the nonce. This attestation step would fail if it detects the phone is rooted. We only managed to execute this attack in an older Samsung phone running Android 8.0.

Extracting the signing key. We discovered a flaw that allowed us to extract the signing key itself.[1] This then allowed us to run the entire location reporting protocol outside the

[1] We decide to withhold the details of this flaw as it remains unpatched, to avoid it being exploited.

phone as we now possess all the information to pass the authentication stage to get the access token.

4.3 Service Operator (A3-RQ3)

Attack Scenario Without a strong end-to-end privacy, the service operator may infringe user privacy by inferring social connections through location history analysis.

End-to-end privacy The OF protocol assumes the vendor as the trusted party, since the vendor has the key material needed to compute the privacy IDs for any registered SmartTag. This means that the vendor can de-anonymize the location reports of a lost device. From a privacy standpoint, this is worse than Apple's FindMy network, in which the cryptographic key needed to generate the privacy IDs is not disclosed to the vendor.

A more interesting question is whether the design of Samsung OF protocol protects the privacy of helper devices. From the location reporting protocol in §3.3, we see that the access token can be used to link multiple location reports. Since each access token is assigned to a helper device, its location reports can be linked to plot its trajectory over the validity period of the token. In principle, the access token is not tied to a particular Samsung account. However, under normal operations, such a token request would be accompanied by other requests to Samsung servers that they can potentially be correlated.

4.4 Tag Owner (A4-RQ2)

Attack Scenario Without adequate anti-tracking mechanisms, malicious SmartTag owners could misuse the OF network for stalking purposes, e.g., covertly track a colleague by hiding the tag in their belongings, or create a hard-to-detect customized tracker leveraging Samsung's OF protocol.

OF device emulation. Emulating an OF device helps both in understanding the various sub-protocols involved in the interactions between the devices and vendor's server and in evaluating the feasibility of creating custom trackers to evade Samsung's anti-tracking mechanisms. Emulating FMM mobile devices is straightforward as the (only) secret for generating privacy IDs can be extracted easily from the device log. For SmartTags, this is not possible. To impersonate a SmartTag that can be registered through normal flow, the impersonation would need to successfully pass the finalization stage (see §3.1), to establish a shared secret with the server. Since the private key for deriving the secret is embedded in the hardware of a legitimate tag and is not explicitly exchanged during registration, it cannot be obtained easily without performing a hardware-level attack on the tag. Unlike the case with FMM device, this shared secret is not recorded in the device log of the owner device. However, we managed to obtain the shared secret by setting up a MITM attack between the owner device and the server, and monitor their exchanges. Since this shared secret is sent by the server to the owner device, we sidestep the need for extracting the private-public key of the tag.

Anti tracking feature The FMM version 7.2.25.14 introduces an anti-tracking module for background detection of tracking tags. The OF data advertised by a lost mode SmartTag contains two temporary identifiers: the Non-Resolvable RPA it uses and the privacy ID in the BLE payload. For a lost mode tag, the two change every 15 minutes. For an overmature lost mode tag, the MAC address still randomizes as normal, yet the privacy ID only updates once every 24 hours.

We noticed that the anti-tracking feature could only detect SmartTags in overmature lost mode through initial observations. Reverse-engineering of the FMM app has shown that the tracking detection algorithm uses the privacy ID contained in the BLE advertisement data of a tag as its identifier, then uses two thresholds to determine whether the tag is a tracker: (1) the duration since the tag was first discovered and saved to the local database; (2) the distance traveled while the tag is nearby (according to the geolocations saved to the database). This explains why the algorithm cannot detect tracking tags in lost mode: Since the anti-tracking algorithm uses the privacy ID data as the only identifier of a tag, it cannot correlate a tag before and after its privacy ID changes. Thus, the detected tracking duration is at most 15 minutes for an lost mode tag. While, for an overmature lost mode tag, the privacy ID value only changes once a day, which typically allows both thresholds to eventually satisfy. Another consequence of this is that privacy IDs generated by offline FMM mobile devices are also ignored by the anti-tracking mechanism.

The above allows an attacker to circumvent anti-tracking through a custom BLE tag that either impersonates an offline FMM device, or a SmartTag with fast rotating privacy IDs.

4.5 Updates on the vendor bug fixes

1. Small Privacy ID Pool (§4.1): The pool size remains the same, but an obfuscation process has been introduced to make the privacy IDs appear more unique. However, attackers can de-obfuscate these IDs (Appendix B.2.3).

2. Linkability Flaws (§4.1): In the latest firmware version (1.04.01), the `IDENTIFIER` characteristic only returns the tag's serial number to authenticated devices, the `HASHED_SN` is no longer writable, and the silent pairing (with tag) vulnerability has been fixed. However, the DFU service remains vulnerable.

3. Location Report Protocol: Samsung has implemented a consistency check to secure the nonce used for access token renewal, preventing the signature replay attack from §4.2. Yet, the other two location report vulnerabilities remain.

5 Discussions

We now discuss a broader implication of our findings, in terms of what we think are issues that go beyond the specific implementation we presented in the preceding sections. Again, our discussion here is organised around the four research questions we pose in the introduction, which we think highlight important features of an offline location tracking system. We shall contrast Samsung's OF protocol design and implementation against Apple's Find My (as analyzed in [21]).

Both Apple and Samsung seem to strike a balance in providing privacy to the owner of a tracker (RQ1) while at the same time providing means to detect unwanted tracking (RQ2). The latter seems to be a focus of much public attention, as indicated by articles in major news outlets such as [12], and has prompted an initiative [23] by Apple and Google to jointly develop a standard to ensure that future BLE trackers will have features built-in that allow cross-platform detection of unwanted tracking. On the issue of unlinkability of BLE data, our conclusion, given our findings and many in BLE fingerprinting work [8–10, 24], it seems quite impossible to design a system that would fully enforce unlinkability.

On the issues prompted by RQ2 (unwanted tracking), both Apple and Samsung seem to adopt a similar approach: a lost tracker will transition to a "lost mode", under which its privacy ID rotates infrequently, allowing an easy detection by a victim's phone. The essential difference in their implementations is in terms of how long it takes for a lost tracker to transition into the lost mode. In Samsung's case, it takes 24 hours, which provides ample time for the attacker to achieve its target. However, both Apple [25] and Samsung designs share a similar flaw: their anti-stalking algorithms seem to ignore BLE devices which are not dedicated trackers.

Samsung's protocol design follows a fundamentally different approach to Apple's with respect to end-to-end privacy (RQ3). Apple's design follows a *decentralised* approach, where the cryptographic keys controlling the generation of privacy IDs are controlled by the end users. This guarantees, in principle at least, that Apple cannot de-anonymize a privacy ID without additional information. This is in contrast to Samsung's *centralised* approach, where the cryptographic keys for generating privacy IDs are known and controlled by Samsung server. This fact may come as a surprise to the reader, and as far as we know, it is not something that is well-understood by the public. This centralised approach does, however, have an advantage over the decentralised approach when it comes to identifying a stalking tag. In the case of Apple's AirTags, this identification is done through reading its serial number, which can be linked to the owner's account. But this assumes the victim can locate the tag and access it physically, and that the tag is a genuine AirTag, so not a custom-modified tag (e.g., using the Open Haystack framework [30]). In the case of Samsung's OF system, the server possesses the information to identify a stalking tag through its privacy IDs, without needing physical access to the tag. From a law enforcement perspective, this allows an easier attribution of the attack.

Lastly, concerning the location report integrity (RQ4), there is an inherent difficulty in preventing the relay attack that is not specific to offline location tracking systems. This issue, for example, also manifests in BLE-based contact-tracing systems. A potential solution could be to adopt some sort of distance-bounding protocols [6] to ensure physical proximity, but whether such an approach is practical and whether it will not introduce further vulnerabilities into the system, is something that is beyond the scope of the current paper.

6 Conclusion

In this work, the Offline Finding (OF) and device management protocols for Find My Mobile (FMM) devices and SmartTags have been thoroughly analyzed, and a security and privacy analysis was performed. Our analysis of the protocols' design and implementation has identified several flaws, allowing each of the research questions to be answered definitively.

We have also discovered vulnerabilities outside the scope defined by the proposed research questions, including multiple other flaws related to the GATT server implementation for SmartTags, and the flaw in the registration protocol that allows an attacker to register a SmartTag of someone else without knowing its ECDH private key (Appendix D.2 on arXiv).

Most of the flaws we identified have been fixed in the latest firmware, so some of our findings and analysis results may not apply to devices/tags with higher version numbers. However, our tests show that devices or tags with older firmware/software versions can still participate in the OF network. Existing users of SmartTags and other FMM devices who have the option to upgrade the firmware/apps on their devices are encouraged to do so to mitigate the issues we discuss here.

Among the issues discussed, of great concern is the possibility of unwanted tracking using SmartTags and similar trackers, such as AirTags and Tile, or custom trackers leveraging on these offline finding networks. The current fragmented approach to anti-stalking features leaves a significant number of people vulnerable to unwanted tracking without an effective mean for detecting it. Fortunately, a standardisation effort is on-going, by Apple and Google, to allow a cross-platform detection of unwanted tracking [23]. We hope our analysis would help inform the design choices in such a process. For future work, we plan to investigate ways to detect unwanted tracking that are effective against a variety of OF networks, leveraging on existing efforts such as AirGuard [20].

Acknowledgments

Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595).

References

[1] Android. Android security bulletin—november 2020. https://source.android.com/security/bulletin/2020-11-01, 2020. Accessed 06-Feb-2023.

[2] Apple. Apple findmy. https://support.apple.com/find-my, 2023. Accessed 06-Feb-2023.

[3] Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, Jonas Höchst, Joshua Kühlberg, Mira Mezini, Richard Mitev, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, Alvar Penning, Dermot Frederik Pustelnik, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. Mind the GAP: security & privacy risks of contact tracing apps. In Guojun Wang, Ryan K. L. Ko, Md. Zakirul Alam Bhuiyan, and Yi Pan, editors, *19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020, Guangzhou, China, December 29, 2020 - January 1, 2021*, pages 458–467. IEEE, 2020.

[4] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[5] Luca Bongiorni. Samsung smarttag hack. https://github.com/whid-injector/Samsung-SmartTag-Hack, 2021.

[6] Stefan Brands and David Chaum. Distance-bounding protocols (extended abstract). In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 1993.

[7] Zak Brighton-Knight, Jim Mussared, and Alwen Tiu. Linkability of rolling proximity identifiers in google's implementation of the exposure notification system. https://github.com/alwentiu/contact-tracing-research/blob/main/GAEN.pdf, 2021.

[8] Guillaume Celosia and Mathieu Cunche. Fingerprinting bluetooth-low-energy devices based on the generic attribute profile. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things, IoT S&P@CCS 2019, London, UK, November 15, 2019*, pages 24–31. ACM, 2019.

[9] Guillaume Celosia and Mathieu Cunche. Saving private addresses: an analysis of privacy issues in the bluetooth-low-energy advertising mechanism. In *MobiQuitous 2019, Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Houston, Texas, USA, November 12-14, 2019*, pages 444–453. ACM, 2019.

[10] Guillaume Celosia and Mathieu Cunche. Discontinued privacy: Personal data leaks in apple bluetooth-low-energy continuity protocols. *Proc. Priv. Enhancing Technol.*, 2020(1):26–46, 2020.

[11] Char49. Samsung Find My Mobile vulnerability. https://char49.com/tech-reports/fmmx1-report.pdf, 2019. Accessed 01-Oct-2022.

[12] James Clayton and Jasmin Dyer. Apple AirTags - A perfect tool for stalking. https://www.bbc.com/news/technology-60004257, 2022. Accessed 01-Oct-2022.

[13] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[14] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[15] Google. Google find my device. https://www.google.com/android/find/, 2023. Accessed 06-Feb-2023.

[16] Gooogle and Apple. Privacy-preserving contact tracing. https://covid19.apple.com/contacttracing, 2020.

[17] Core Specification Working Group. Bluetooth core specification v5.3. https://www.bluetooth.com/specifications/specs/core-specification-5-3/, July 2021. Accessed 06-Feb-2023.

[18] Samsung Group. Samsung SmartThings Find hits new milestone with 200 million nodes helping find lost devices. https://news.samsung.com/global/samsung-smartthings-find-hits-new-milestone-with-200-million-nodes-helping-find-lost-devices, 2022. Accessed: 2022-07-28.

[19] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[20] Alexander Heinrich, Niklas Bittner, and Matthias Hollick. Airguard - protecting android users from stalking attacks by apple find my devices. *CoRR*, abs/2202.11813, 2022.

[21] 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. *Proc. Priv. Enhancing Technol.*, 2021(3):227–245, 2021.

[22] Vincenzo Iovino, Serge Vaudenay, and Martin Vuagnoux. On the effectiveness of time travel to inject COVID-19 alerts. *IACR Cryptol. ePrint Arch.*, page 1393, 2020.

[23] Brent Ledvina, Zachary Eddinger, Ben Detwiler, and Siddika Parlak Polatkan. Detecting Unwanted Location Trackers. Internet-Draft draft-detecting-unwanted-location-trackers-00, Internet Engineering Task Force, May 2023. Work in Progress.

[24] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik C. Rye, Brandon Sipes, and Sam Teplov. Handoff all your privacy - A review of apple's bluetooth low energy continuity protocol. *Proc. Priv. Enhancing Technol.*, 2019(4):34–53, 2019.

[25] Travis Mayberry, Ellis Fenske, Dane Brown, Jeremy Martin, Christine Fossaceca, Erik C. Rye, Sam Teplov, and Lucas Foppe. Who tracks the trackers?: Circumventing apple's anti-tracking alerts in the find my network. In *WPES '21: Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society, Virtual Event, Korea, 15 November 2021*, pages 181–186. ACM, 2021.

[26] RFC5652. Cryptographic Message Syntax (CMS). https://www.rfc-editor.org/rfc/rfc5652, 2009. Accessed 01-Oct-2022.

[27] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. Airtag of the clones: Shenanigans with liberated item finders. In *15th USENIX Workshop on Offensive Technologies, WOOT 2022*. USENIX, 2022.

[28] Samsung. Samsung find my mobile. https://findmymobile.samsung.com/, 2023. Accessed 06-Feb-2023.

[29] Samsung. Samsung find my mobile app. https://www.samsung.com/au/apps/find-my-mobile/, 2023. Accessed 06-Feb-2023.

[30] SEEMOO. Open haystack. https://github.com/seemoo-lab/openhaystack, 2021.

[31] Nordic Semiconductor. nRF5 SDK v17.0.2. https://infocenter.nordicsemi.com/topic/sdk_nrf5_v17.0.2/index.html, 2020. Accessed 07-Feb-2022.

[32] Alwen Tiu and Jim Mussared. A silent pairing issue in Bluetooth-based contact tracing apps. https://github.com/alwentiu/COVIDSafe-CVE-2020-12856/blob/master/CVE-2020-12856-19-June-2020.pdf, 2020.

[33] Carmela Troncoso, Dan Bogdanov, Edouard Bugnion, Sylvain Chatel, Cas Cremers, Seda F. Gürses, Jean-Pierre Hubaux, Dennis Jackson, James R. Larus, Wouter Lueks, Rui Oliveira, Mathias Payer, Bart Preneel, Apostolos Pyrgelis, Marcel Salathé, Theresa Stadler, and Michael Veale. Deploying decentralized, privacy-preserving proximity tracing. *Commun. ACM*, 65(9):48–57, 2022.

[34] Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. Lost and found: stopping bluetooth finders from leaking private information. In *WiSec '20: 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Linz, Austria, July 8-10, 2020*, pages 184–194. ACM, 2020.

[35] Wikipedia. Samsung galaxy smarttag. https://en.wikipedia.org/wiki/Samsung_Galaxy_SmartTag, 2023. Accessed 06-Feb-2023.

[36] Fenghao Xu, Wenrui Diao, Zhou Li, Jiongyi Chen, and Kehuan Zhang. Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[37] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking secure pairing of bluetooth low energy using downgrade attacks. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 37–54. USENIX Association, 2020.

A Table of Acronyms

Acronym	Full Name
FMM	Find My Mobile
Helper Device	A device that discovers and reports lost FMM devices/SmartTags
Lost Device	An FMM device/a SmartTag operating in lost mode or overmature lost mode
Owner Device	A device signed in with a Samsung account that owns FMM device(s)/SmartTag(s)
OF	Offline Finding
Privacy ID	A unique identifier of an FMM device/a SmartTag

Table 3: Samsung's Offline Finding Protocol Acronyms

Acronym	Full Name
AES	Advanced Encryption Standard
API	Application Programming Interface
APK	Android application Package
BLE	Bluetooth Low Energy
CA	Certificate Authority
DFU	Device Firmware Update
ECDH	Elliptic-curve Diffie–Hellman
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GUI	Graphical User Interface
IRK	Identity Resolving Key
LTK	Long Term Key
MAC	Media Access Control
MITM	Man-in-the-Middle
RPA	Random private address
TLS	Transport Layer Security
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

Table 4: Common Acronyms

B Offline Finding protocol for smartphones

In this section, we discuss our findings in the reverse engineering of offline finding features of the FMM app. The results of this section apply to all versions of FMM app (with the offline finding features) prior to version 7.2.24.12 (July 2022). We have not studied comprehensively the patched FMM app, but some preliminary findings related to how the advertisement payload is generated is given in Appendix B.2.3.

The OF protocol has multiple modes of operations that depend on the functions supported by the devices involved. We outline the main OF protocol that applies to mobile devices, which consists of four main operations: Device/Account Registration, lost (offline) device Operation, helper (online) device Operation, and Device/Account Deregistration. To simplify presentation, we omit the detailed concrete details of various messages exchanged and would refer to some important data symbolically. We also conflate the different servers used in the OF protocol to a single entity, which we simply refer to as "the server".

The protocol can be summarised as follows. Initially, devices must complete the registration process with the server to obtain various parameters that will be used in the offline operation. When a registered device goes offline, it starts advertising a unique payload that identifies itself. This payload is picked up by nearby online (registered) devices which parse the payload extracting the device's identifier. The online device then accesses available location services to find out its own location. It then sends the lost device's identifier and the location through to the server. The owner of the lost device can then access the server to find out its location. Further details of each operation are outlined below.

B.1 Device registration

The Offline Finding (OF) feature can be enabled on Samsung smartphones in the device settings when the device is signed in with a Samsung account. The device will then make an HTTPs request to the `/v1/kms/cf/device/registerDevice` URL of the `samsungdive` server to register itself to the OF network. The request contains the following information:

- Secret key: 16-byte random generated by Java Random.
- Device ID: Base64 encoding of an MD5 hash of the device's IMEI, which is constant and unique for a device.
- User ID: A value associated with the Samsung account logged in.
- Other information such as device type, region code, client version, and device model.

The server then responds with several pieces of information; the most important one (from a security/privacy perspective) is the `PrivateIDConfig`, which consists of a 16-byte secret key, an IV (`PrivacyIV`), and a size parameter (`PrivacyPoolSize`) that will be used to generate BLE advertisements for the OF operations. The IV value is fixed to the same 16-byte value (i.e., `"+IABCfvBZHJYFUek8vp3Gg=="` in base64 encoding) across accounts and devices. The privacy ID pool size determines the amount of possible advertising values the device will generate. This has also been observed to be standard for all devices, taking the value 51. This means that there are only 51 possible advertisement payloads from a lost device.

Registration process details Listing 1 contains the system log produced when enabling the Offline Finding Service on a phone. The Highlighted parts show the information for the device registration request and response. The request contains the account and device information, such as `userId`, `deviceId`, and `email`. The server's response contains the Offline Finding policy, the `privateIdConfig`, and other configuration data.

Listing 1: FMM device registration log

```
10–07 17:35:33.429 8983 21707 I DBG_FMMSEC :[AccountManager ]:
      getAccountInfo from preference : AccountVO{userId='***',
      deviceId='IMEI:', cntCode='AUS', email='***',
      serverUrl='www.ospserver.net', mcc='505', remoteControl=true}
10–07 17:35:33.430 8983 21707 I DBG_FMESEC :[CFHTTPRequestor ]: request
      url : https://eu-kms.samsungdive.com/v1/kms/cf/device/registerDevice
10–07 17:35:56.916 8983 21707 I DBG_FMESEC :[CFHTTPRequestor ]: Reqeust
      result :
  {
    "policy":{
      "version":"1",
```

```
        "type":"mobile",
        "advertiseInterval":10000,
        ...
    },
    "TargetURL":"***",
    "privateIdConfig":{
        "secretKey":"***",
        "iv":"+IABCfvBZHJYFUek8vp3Gg==",
        "privacyIdPoolSize":51
    },
    "responseCode":200
}
```


B.2 Lost device operation

When a OF registered device no longer has an active network connection, it enters 'Lost Mode' and triggers the Offline Finding service to start. The lost device then creates a GATT server profile and starts advertising on the main OF service UUID (FD69). The advertising is the fundamental operation for lost devices as part of the main OF protocol. The GATT server is not directly used by the main OF protocol, however it can be interacted with via BLE and is used as part of secondary OF protocols.

B.2.1 GATT server profile

Table 5: FMM GATT Services

Name	UUID
ENCRYPTION	EEDD5E73-6AA8-4673-8219-398A489DA87C
FME	4EBE81F6-B952-465E-9ECE-5CA39D4E8955

The GATT profile of a lost device contains two services: the encryption service and the FME service, as shown in Table 5.

- **Encryption Service** (`ENCRYPTION`). This service is used to implement a challenge-response protocol for authenticating a device that wants to connect to the lost device. It contains three characteristics (see Table 6): The `SUPPORTED_CIPHER` characteristic is readable and contains information about the cipher to be used, which is AES/CBC/PKCS7. The `NONCE` characteristic is readable and returns an IV to be used during encryption. This IV is a random nonce that is generated using Java `SecureRandom` each time a client connects to the server. The `ENONCE` characteristic is writeable and expects to receive an encrypted version of the string "smartthings". This string must be encrypted using the given IV and with the device's secret key (from the `PrivateIDConfig`). Writing the correct ciphertext to the encrypted nonce characteristic completes the handshake between client and server.

Table 6: Characteristics under the Encryption Service

Name	UUID
NONCE	A12BE31C-5B38-4773-9B9D-3D5735233A7C
ENONCE	4EBE81F6-B952-465E-9ECE-5CA39D4E8955
SUPPORTED_CIPHER	50F98BFD-158C-4EFA-ADD4-0A70C2F5DF5D

- **FME Service** (`FME`). After completing the handshake, the client is now authenticated and can interact with the characteristics in the FME service. The device's alarm can be set to ring by writing the byte 01 (encrypted using the same cipher) to the `ALARM` characteristic (see Table 7).

Table 7: Characteristics under the FME Service

Name	UUID
ALARM	4a1351bb-d617-4612-a8e3-8dee6ca13e7b
CCCD	00002902-0000-1000-8000-00805f9034f0
MCF	0487d871-d55e-44aa-8318-4faa721278e5

B.2.2 BLE operations

Privacy ID generation The lost mode advertisements are the fundamental part of the OF protocol. The lost device generates an advertisement containing a unique identifying payload which is picked up by a helper and reported to Samsung. A key component of the advertisement payload is the *privacy ID* that identifies the device uniquely. A device can generate a number of privacy IDs, depending on the privacy ID pool parameter in `PrivateIDConfig`.

Let k, iv and p denote, respectively, the secret key, IV and the privacy ID pool from the device's `PrivateIDConfig`. To generate a privacy ID, first we compute a 20-byte array:

$x_i =$

Byte 0	Byte 1	Bytes 2-17	Byte 18	Byte 19
00	i	k	00	i

where i is a 1-byte random nonce in $\{1,\ldots,p\}$, i.e., it is a random 1-byte value bounded by the privacy ID pool. From x_i, one then generates a ciphertext: $y_i = E_k(iv, x_i)$, where E denotes the AES/CBC/PKCS7 cipher, initialized with the key k and the initialization vector iv. The privacy ID corresponding to each i is then computed by taking the first 12 bytes of y_i: $pid_i = y_i[0:11]$.

BLE advertisement generation Finally, the advertisement payload is generated from the privacy ID combined with various meta data. Table 8 describes the full advertisement payload; Table 9 provides details of the support info byte.

Table 8: FMM lost mode advertisement structure

Byte	0	1-12	13
Info	operation mode	privacy ID	support info

The first byte describes the operation mode of the OF protocol that is being used by a lost device. In the main OF protocol,

Table 9: Support info byte (byte 13)

Bit	0-3	4	5	6	7
Info	region info	E2E flag	UWB flag	MCF flag	reserved

this byte is always zero. The last byte contains information about the device's region and functionalities supported. This last byte varies depending on the device but stays consistent for all advertisements for a device. If two lost mode devices are advertising in the same area, then this last byte can be used as a quasi-differentiator between the two, provided they do not have the same settings/support.

Once the advertising data has been generated, the lost device starts advertising over BLE on the OF service's UUID FD69. The device will continuously advertise the same data until a timer is triggered that causes it to shuffle the advertising data. This timer is set to trigger every 60 minutes, after which the device generates a new random nonce to be used to generate the advertising data. Since, there are only 51 possible values for the nonce i, and it is the only source of non-determinism, there are also only 51 possible values of the advertising data (for a `PrivateIDConfig`). The lost device repeats this process until it is online again. If an adversary has access to the device's secret key and IV, then it is trivial to generate the 51 possible values.

B.2.3 BLE operations for newer FMM versions

Privacy ID obfuscation Samsung has introduced change to the lost mode advertisement format for Galaxy smart devices with FMM version 7.2.24.12 or above.

Table 10: Payload format for new FMM lost mode advertising

Byte	0	1-12	13	14	15-17	18-19
Info	operating mode	(obfuscated) privacy ID	support info	reserved	aging counter	signature

As shown in Table 10, the key differences for the new FMM advertisement structure include the following:

obfuscated privacy ID an obfuscation method is applied to the privacy ID contained in the BLE data. Details of the obfuscation method will be explained next.

timestamp the added timestamp field is used to store a 3-byte value that represents the time when the advertisement data was computed. It is computed by dividing the current system time by 900 and casting the result to an integer:

$$timestamp = currentTimeSeconds//900$$

signature the added signature field stores a 2-byte value. It is computed using `HMAC-SHA256` with the device's secret key, and first 18 bytes of the advertisement data.

It was observed that the new FMM version still uses a size-51 privacy ID pool as for the older versions, meaning that there will only use 51 unique raw privacy ID for each device. However, an obfuscation algorithm is used to add more randomness to the privacy ID value contained in the advertisement data.

The algorithm uses a universally static obfuscation table and deterministically generated random Bytes to obfuscate each raw privacy ID:

1. First, the FMM app computes a 12-byte raw privacy ID using the same algorithm as in older versions
2. Then, the app computes the current timestamp value and saves it to a variable, which is later used to determine the obfuscation filter being used and the random bytes generated by `java.util.Random`.
3. The obfuscation table is an `ArrayList<String>` object that contains four hex strings as shown in Table 11. For each advertisement data generation process, the FMM app selects the $timestamp \mod 4$th string and convert it to a 12-byte filter.

Table 11: Content of the obfuscation table

Index	Value
0	88DFAF0581FFCEB1429F2200
1	4A2635F7AD0E416906A35CBE
2	19DB724B07DF72B9792511DE
3	7BC79BAB386B8AFEFE63B9B7

4. Then, the app creates a new instance of the `Random` class and uses the $timestamp$ as the seed to generate 12 random bytes deterministically.
5. Finally, the obfuscated privacy ID is computed as follows:

$$privacyID_{obf} = privacyID_{raw} \oplus filter \oplus randomBytes$$

Privacy ID de-obfuscation The obfuscation operation can be easily reversed by an adversary to extract the raw privacy ID from the lost advertisement data observed over BLE:

1. The adversary uses the $timestamp$ value in the BLE data as the seed for `java.util.Random`, which allows $randomBytes'$, the same random bytes used in the obfuscation process, to be generated.
2. Then, the adversary selects $timestamp \mod 4$th string from the universally static obfuscation table as the filter, $filter'$.
3. Finally, the adversary can obtain the raw privacy ID from the following XOR operation:

$$privacyID_{raw} = privacyID_{obf} \oplus filter' \oplus randomBytes'$$