Toward a Secure Crowdsourced Location Tracking System

Chinmay Garg
University of California, Santa Barbara
chinmay@ucsb.edu

Aravind Machiry
Purdue University
amachiry@purdue.edu

Andrea Continella
University of Twente
acontinella@iseclab.org

Christopher Kruegel
University of California, Santa Barbara
chris@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
vigna@cs.ucsb.edu

ABSTRACT

Low-energy Bluetooth devices have become ubiquitous and widely used for different applications. Among these, Bluetooth trackers are becoming popular as they allow users to track the location of their physical objects. To do so, Bluetooth trackers are often built-in within other commercial products connected to a larger crowdsourced tracking system. Such a system, however, can pose a threat to the security and privacy of the users, for instance, by revealing the location of a user's valuable object. In this paper, we introduce a set of security properties and investigate the state of commercial crowdsourced tracking systems, which present common design flaws that make them insecure. Leveraging the results of our investigation, we propose a new design for a secure crowdsourced tracking system (SECrow), which allows devices to leverage the benefits of the crowdsourced model without sacrificing security and privacy. Our preliminary evaluation shows that SECrow is a practical, secure, and effective crowdsourced tracking solution.

CCS CONCEPTS

• **Security and privacy → Software reverse engineering**; **Information flow control**; **Public key encryption**; *Tamper-proof and tamper-resistant designs*; *Penetration testing*.

ACM Reference Format:
Chinmay Garg, Aravind Machiry, Andrea Continella, Christopher Kruegel, and Giovanni Vigna. 2021. Toward a Secure Crowdsourced Location Tracking System. In *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28-July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3448300.3467821

1 INTRODUCTION

Tracking devices have become very common for a wide range of applications and use cases. People use tracking devices to monitor trucking fleet movements, commercial and defense vehicles, monitor their pets and keep track of the location of their valuables in case they are stolen or lost. While trackers can rely on different technologies, such as radio waves, GPS, and WiFi, Bluetooth Low-Energy (BLE) trackers have become the main choice for consumer-targeted trackers [15, 58, 62], and they are currently used by millions of users. For instance, the popular TrackR app has more than 1,000,000 downloads from the Google Play Store [37].

In practice, BLE trackers consist of small and cheap devices that can be attached to key rings, luggage, wallets, or any other personal belonging that a user wishes to track. Once activated, they work by communicating with a mobile app installed on the users' smartphone. The app allows the users to interact with the trackers (e.g., make it ring) and acts as a medium of communication to "connect" the BLE trackers to the Internet. In fact, the "killer feature" of such devices is the ability for users to remotely query for the location of their valuables. To do so, BLE trackers are embedded into a more complex ecosystem, called *crowdsourced location tracking system.* In a crowdsourced model, users can track the location of their physical devices by leveraging information collected and produced by *other* users in the network. In practice, crowdsourced tracking systems rely on all their users, and on their smartphones' GPS, to obtain information about the location of their nearby BLE trackers. This allows the owner of a BLE tracker to know the location of their valuable (almost) in real-time, even when they are not in proximity of the tracker. At the same time, the crowdsourced model allows vendors to drastically reduce the cost of their trackers, as they do not need to embed GPS capabilities.

However, due to the nature of the distributed crowdsourced model, which utilizes smartphones for location updates, crowdsourced location tracking systems raise questions regarding the security and privacy of their protocols. Such questions come naturally since attackers could potentially abuse the systems to perform illicit operations, *e.g.*, stealing or inferring the location of a valuable object. Indeed, a recent study showed that different tracking systems led to various privacy issues [66].

In this work, we perform a detailed, principled, and comprehensive security analysis of the existing crowdsourced location tracking systems. We first formalize a set of security properties that every crowdsourced tracking system should fulfill, together with the conditions that must be satisfied to guarantee such properties. We then study the most popular, commercial, crowdsourced tracking systems, identifying major systematic flaws that violate our security properties and, therefore, allow attackers to obtain unauthorized access to tracked devices. We responsibly disclosed our findings to the affected vendors.

Finally, inspired by our security properties, we design the protocol of a secure crowdsourced tracking system (SECrow), which allows users to leverage all the benefits of the crowdsourced model without compromising their security and privacy. Our design guarantees reliable (i.e., unspoofable) and anonymous location information, and provides end-to-end encryption, where only the owner of a BLE tracker

WiSec '21, June 28-July 2, 2021, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-8349-3/21/06.
https://doi.org/10.1145/3448300.3467821

can access its location. As a consequence, our design protects the location information even from server-side data breaches [16, 59, 70].

In summary, we make the following contributions:

- We investigate the security issues arising from the adoption of the crowdsourced model for location tracking. We define and formalize a set of security properties and conditions that crowdsourced location tracking systems must guarantee.
- We perform a security analysis of the most popular, commercial, crowdsourced tracking systems, demonstrating a real threat for users. Specifically, by analyzing and instrumenting their mobile apps, we identify major design flaws in all these systems. We reported our findings to the responsible entities.
- We design SECrow, a secure protocol for crowdsourced location tracking that fulfills all our security properties, protecting users from various attacks, and providing end-to-end encryption. We show the practicality of our approach by implementing and testing a prototype.

In the spirit of open science, we make all our code available at https://github.com/ucsb-seclab/SECrow.

2 BACKGROUND

In this section, we explain crowdsourced tracking using common terms that will be used throughout the paper. The goal of crowdsourced tracking is to provide users with the ability to track the location of physical devices by leveraging the crowdsourcing model, *i.e.*, by leveraging information collected and produced by other users in the network. A typical crowdsourced tracking system involves three entities (Figure 1).

- **Tracking Devices (TDs):** The actual devices being tracked (*e.g.*, TrackR [62], CUBE [18]). These devices are battery-powered, communicate using Bluetooth Low Energy (BLE), and they are compact so that they can be attached to other physical objects, such as wallets and keys. This provides the ability to track the corresponding physical objects by leveraging the attached TD.
- **Tracking Service (TS):** A web service (usually REST-based [45]) provided by the manufacturer of TDs that allows users to know the location of their TDs.
- **Communication Device (CD):** A device that can directly communicate with a TD. Most commonly, this is an Internet-enabled smartphone running an app, which we call Tracking App (TApp), provided by the TD vendors. The TApp can communicate with the TS and TDs. If the CD has location service capabilities, then the TApp can update TS with the current location of all the BLE-visible TDs, *i.e.*, those that are nearby. This information from CDs represents the crowd input. The information from all the CDs is used by the TS to provide the ability for users to know the location of their TDs. The TApp also provides an easy way for users to track TDs and thus provides an implicit incentive for the users to install the app, thereby participating in crowdsourced tracking.

Owners. Every TD is owned by one or more users who can perform privileged operations on it, such as make it ring and flash. The ownership is established by registering and claiming the device through a pairing process, where the TApp adds the TD as a registered item within the user account. Generally, users use TApp to interact with owned TDs through BLE. Furthermore, owners can query the last known location of a TD, that is *lost or not in BLE proximity*, by communicating with the TS—through the TApp.

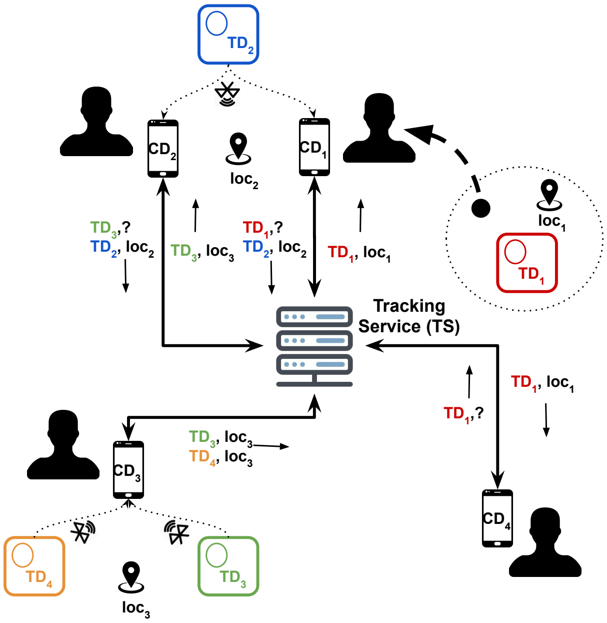

Figure 1: Data-flow of a representative crowdsourced tracking system. CD$_1$ and CD$_4$ own TD$_1$. CD$_2$ owns TD$_2$ and TD$_3$. CD$_3$ owns TD$_4$. The dashed arrow indicates that the user with CD$_1$ moved from location loc_1 to loc_2.

A representative crowdsourced tracking system is depicted in Figure 1, which shows four TDs, four CDs, and the TS: CD$_1$ and CD$_4$ own TD$_1$; CD$_2$ owns TD$_2$ and TD$_3$; CD$_3$ owns TD$_4$. The dashed arrow indicates that the user with CD$_1$ moved from location loc_1 to loc_2.

Location update. All CDs constantly update the TS with the location of the TDs that are in BLE proximity. Specifically, CD$_1$ and CD$_2$ update the location of TD$_2$ using the tuple TD$_2$, loc_2. Similarly, CD$_3$ updates the location of TD$_3$ and TD$_4$. CD$_4$ does not send any location update since there are no TDs nearby.

Location query. The owners of a TD can query TS about the location of their TD. The TS responds with the last known location of the TD. In Figure 1, we can see that CD$_1$, CD$_2$, and CD$_4$ issue location queries for TD$_1$, TD$_3$, and TD$_1$ respectively represented by a tuple of the form TD$_x$,?. TS responds to such location queries with the last known location of the TDs in the form TD$_x$, loc_y. CDs do not need to send location queries for their TDs when they are nearby, as they can communicate directly through BLE. For instance, CD$_3$ does not send any location query for TD$_4$ as it is BLE visible to the corresponding CD$_3$.

3 SECURITY PROPERTIES

As already mentioned, the main goal of any crowdsourced tracking system is to provide the ability for users to reliably obtain the location of their tracked devices. From a security and privacy perspective, we also want such systems to guarantee certain properties, *e.g.*, only owners can perform privileged operations. Furthermore, for each property, we identify the *necessary conditions* that need to be satisfied to guarantee the corresponding property. We call these the *Necessary security conditions* (NSCs) of a security property. This can be represented formally as,

$$\forall C \in NSCs(S) : \neg C \rightarrow \neg S.$$

where $NSCs(S)$ is the set of all the necessary conditions for the security property S. This formulation implies that if any of the NSCs does not hold, then the corresponding security property is violated, providing an effective method to check for *violations* of the security properties. All the security properties are named as $X-S_y$, where X is the entity the security property belongs to, and y is a number indicating a unique id. Similarly, all the necessary conditions are named as C_n, where n is a positive number. Next, we list all the security properties and their corresponding necessary conditions.

3.1 TD security properties

Privileged owners ($TD-S_1$). Only owners can perform privileged operations on a TD. Specifically, we consider the following operations as privileged:

- Obtaining the location of the TD from the TS.
- Making the TD ring or blink remotely (*i.e.*, through BLE) to capture human attention.

To enforce the above restrictions, it is important for a TD and the TS to have an ability to *recognize the owners* of the TD to restrict access to the privileged operations. Hence, we see that following as the NSCs for the $TD-S_1$ security property:

- C_1**:** The TD must able to recognize its owners.
- C_2**:** The TS must able to recognize the owners of a given TD.

Physical ownership ($TD - S_2$). It must be impossible to own a tracking device without physical access. This is based on the traditional ownership requirement *i.e.* to own a device, it is important to verify that the device can be accessed physically. To ensure this, we need the ability to verify physical ownership. Consequently, we need the following NSCs for the $TD-S_2$ security property:

- C_3**:** Owning of a TD must involve performing a physical action on the TD.
- C_4**:** Registering as a owner of a TD with the TS must involve communicating with the TD.

We assume that once registered to a (primary) owner, a TD cannot be registered to a new (primary) owner. As we detail in Section 6, we allow primary owners to register secondary owners.

3.2 CD security properties

Anonymous and proximity-aware location update ($CD - S_1$). A CD should be able to *anonymously* update the location of *only* in proximity TDs. This ensures that participating in the crowdsourced system does not pose any privacy risks for the CD and protects against Sybil attacks [65] by ensuring that location can be updated for only those TDs that are in BLE proximity This constrains that all the sybils need to be in BLE proximity of the victim TD. To guarantee this property, we require that a CD communicates with a TD to update its location. Since TDs can communicate only through BLE, such communication ensures proximity. Hence, the NSC is:

- C_5**:** The CD can update the location of a TD if and only if the CD can communicate with the TD.

3.3 TS security properties

Reliable location service ($TS-S_1$). The location information provided by the TS must be reliable, *i.e.*, the retrieved location of a TD must not be spoofable. This ensures that the location information about a TD provided by the TS can be trusted by the owners.

To have the $TS-S_1$ property, all the entities involved in the location update of a TD (*i.e.*, TD, location information from CD, and, TS) should not be affected. Which results in the following NSCs:

- C_6**:** It must be impossible to spoof a given TD.
- C_7**:** It must be impossible to spoof a location by a CD.
- C_8**:** It must be impossible to spoof the TS[1].

Table 1 summarizes the aforementioned security properties along with the corresponding necessary conditions (C_1-C_8), which form the NSCs of a secure crowdsourced tracking system.

4 THREAT MODEL

In any crowdsourced tracking system, the TS plays a major role as it is the main entity that receives and provides location information for all the TDs. In our work, we consider TS to have *malicious but cautious* behavior [13]. In practice, on the one hand, we assume that the TS does not intentionally attempt to spoof the location information, as users could easily identify tampered location records. On the other hand, we assume that the TS can actively attempt to collect or learn information about the location of the TDs from incoming reports and queries, therefore affecting the users' privacy.

We assume an external attacker that has full control of the software that runs on a CD, except for the code that is protected through hardware isolation mechanisms, such as ARM TrustZone [49].

We consider the TDs to be a *black box* for the attacker. Specifically, the attacker can only communicate with a TD through BLE. As such, attacks involving physically tampering with the TDs and corresponding hardware attacks are out-of-scope for our work. However, we assume that the attacker can snoop in the BLE traffic and try to mimic a TD. Finally, we do not consider vulnerabilities in communication protocols implementations such as BLE and TLS.

Similarly, software vulnerabilities in the code that runs in the TS and TDs are out-of-scope, and well covered by other research work. However, we assume that the TS can be subject to data breaches, and thus we aim at protecting location information even in such scenarios—guaranteeing end-to-end encryption.

In our threat model, attackers are interested in accessing or spoofing information about the location of users' valuable objects, with the goal of, for instance, tracking and stealing such valuables.

5 SECURITY ANALYSIS

We study five popular crowdsourced tracking systems, TrackR [62], CUBE [18], Chipolo [15], Pebblebee [48], and, Tile [58], looking at whether each system guarantees the aforementioned security properties (Section 3). We do so by verifying if all the NSCs hold. All of the studied systems provide a TApp that users can use to communicate with the TDs and the corresponding TS (Table 2). Users must first create an account by registering with the TS. All communications with the TS then require the user to login and include their session token in every request sent to the TS. We use the term *authenticated*

[1] This can be achieved by ensuring that TS uses Transport Layer Security (TLS) [20].

Table 1: Security properties of an ideal crowdsourced tracking system along with the corresponding necessary conditions.

	Security properties of a crowdsource tracking system			
	Tracking device (TD)		**Communication device (CD)**	**Tracking service (TS)**
	Privileged owners ($TD\text{−}S_1$)	**Physical ownership** ($TD\text{−}S_2$)	**Anonymous & proximity-aware location update** ($CD\text{−}S_1$)	**Reliable location service** ($TS\text{−}S_1$)
Necessary Conditions	• TD should recognize its owners (C_1) • TS should recognize owners of a given TD (C_2)	• Owning a TD should involve performing physical action on it (C_3) • TS should be able to verify proof of physical activity (C_4)	• CD can update the location of a TD if and only if the CD can communicate with the TD (C_5)	• Impossible to spoof a TD (C_6) • Impossible to spoof a location by CD (C_7) • Impossible to spoof the TS (C_8)

REST endpoint to indicate that the endpoint requires the user to authenticate first. Some tracking services allow a TD to have multiple owners so that multiple users can share and track a TD. Note that, there exist other BLE tracking systems, such as GoFinder HL [24] and Nut [61], which, however, do not adopt the crowdsourced model and, hence, are out-of-scope for this study.

5.1 Methodology

Our analysis methodology is based on the following two steps.

Record. We record the messages sent from the TApp to the TD and TS. First, we manually reverse engineer the TApp to identify the functions used to send and receive messages from and to the TS and TD. For instance, the TrackR's TApp uses the `write` method of the `java.io.BufferedWriter` class to send messages to the TS. Next, we setup a dynamic instrumentation environment to hook into the identified functions and record all their messages. Finally, we use the corresponding TApp to register a TD, update its location from a different CD, and retrieve the updated location. Our dynamic instrumentation records all the messages sent through the instrumented functions. We carefully select the appropriate functions to hook so that we get messages in plaintext before they encrypted [12]. Thus, hooking at lower layers (*e.g.*, system calls) might not reveal the content of the messages.

We report the details of the communication endpoints of each tracking system in the Appendix of our full version [64].

Verify. We use recorded messages to test the required security properties. For instance, consider the case of verifying the physical ownership property for CUBE. We purchased two CUBE TDs, $D1$, and $D2$, whose Bluetooth MAC address is respectively `MAC_1` and `MAC_2`. During our recording phase, we record that, when registering a new owner for $D1$, the TApp sends an HTTP POST request to the endpoint `/api/devices?AddCubeWithMac=MAC_1` with the following JSON payload: `{"name":"<NAME>", "mac": "MAC_1"}`.

We can see that the payload includes the MAC address of $D1$ (*i.e.*, `MAC_1`) and we do not see any communication with $D1$, which violates NSC C_4. Now, to verify the physical ownership in CUBE, we replay the request by replacing the MAC address with `MAC_2`. A successful request indicates that we can register a new owner for the TD without physical access, thus violating the physical ownership property ($TD\text{−}S_2$). We follow the same black-box methodology to verify each property for each tracking system.

In the following sections, we describe the results of our analysis of the studied crowdsourced tracking systems, and we show that none of them guarantee the desired security properties.

5.2 TrackR

Owner registration. A user can own a TD by pairing with it through BLE. Furthermore, the user also needs to send a `POST` request to TS to register as an owner of the TD. Specifically, the registration request is sent to the REST endpoint with a JSON payload containing a unique identifier, called `trackerid`, of the TD along with `USERTOKEN`. Here, `USERTOKEN` is the login token of the user, and the `trackerid` of the TD is a string composed of four zeros ("0000") and the reverse of the Bluetooth MAC address of the TD. For instance, for a TD with MAC address 00:1B:44:11:3A:B7, the `trackerid` is 0000b73a-11441b00.

This design *violates* both the ***Privileged owners*** ***(**$TD - S_1$***)*** and the ***Physical ownership*** ***(**$TD - S_2$***)*** properties. In fact, the TD, in the case of TrackR, does not keep track of the owners resulting in missing C_1 NSC. Because of this, an attacker can perform privileged operations, such as ringing and blink, on the TD by sending messages over BLE to the TD. Second, as mentioned above, registering as an owner of a TD requires an attacker to send a POST request with the attacker's login token and the `trackerid` of the TD. However, the `trackerid` of a TD can be easily obtained by observing its MAC address, which is broadcasted over Bluetooth and can be known without physical access to the TD. Thus an attacker can register as an owner of any nearby TD, which violates missing C_3 and C_4 NSCs. Moreover, given that the first three bytes are unique for an organization [32], *i.e.*, TrackR, by knowing a MAC address *an attacker can use the last three bytes to enumerate all possible TrackR TDs MAC addresses and register as an owner for every TrackR TD* without having physical access to the TDs, thus violating *Physical ownership* property.

Location update. Updating the location of a TD to the TS is done by sending a PUT request to the REST interface with a JSON payload containing the `trackerid` of the TD along with location information in the form of longitude and latitude values.

This allows an attacker to update the location information of any TD (given its MAC address) without being physically close to the TD. Note that, updating the location of TD does not require communicating with the TD, consequently missing C_5 NSC. This *violates*

Table 2: Crowdsourced tracking systems considered in our security analysis.

Tracking System	**TApp**	**TS**	**Multi-owner Support**
TrackR [62]	com.phonehalo.itemtracker [37]	platform.thetrackr.com	Yes
CUBE [18]	com.blueskyhomesales.cube [43]	net.cubetracker.com	No
Chipolo [15]	chipolo.net.v3 [14]	api.chipolo.net	No
Pebblebee [48]	com.pebblebee.app.hive3 [35]	api.pebblebee.com	No
Tile [58]	com.thetileapp.tile [36]	production.tile-api.com	No

*the **Anonymous and proximity-aware location update** ($CD-S_1$) property.*

Location query. A user can obtain the location of a TD by sending a GET request to the REST interface using the login token of the user. The TS responds with location information of all the TDs owned by the user. Given a TD, an attacker can affect the TD's location information either by updating its location using the method described in Section 5.2 or *by spoofing the MAC address of the TD* (*i.e.*, missing C_6 NSC). Moreover, the TS blindly trusts the location information about a TD provided by any user without verifying the accuracy of the location information missing C_7 NSC. These attacks clearly *violate the **Reliable location service** ($TS-S_1$) property.*

5.3 CUBE

Owner registration. A new owner can register by pairing with the TD and sending a POST request to the TS, through an authenticated REST endpoint. The payload of this request contains the MAC address of the TD along with other data items.

The TS allows a single owner to a given TD. Hence, the TS allows a user to register as the owner of a TD only if the TD is not already registered by another user. However, the TS does not verify that the user has physical access to the TD. This enables an attacker to own an unregistered TD by knowing its MAC address. This *violates the **Physical ownership** ($TD-S_2$) property.* Furthermore, as mentioned in Section 5.2, from a single MAC address of a CD an attacker can enumerate all the possible MAC address of the TDs and register as the owner before a valid user attempts to register.

Similarly to TrackR, the TDs in CUBE do not keep track of their owner (*i.e.*, missing C_1). This allows an attacker to perform privileged operations, such as ringing and blink, on the TD, thus *violating the **Privileged owners** ($TD-S_1$) property.*

Location update. In CUBE, location updates by a user are accepted for a TDs only if either the TD is owned by the same user or the TD is marked as lost by their respective owners. The location update of a TD is sent to the TS using a PUT request with a JSON body containing the device MAC address and location information.

Similar to TrackR, the CUBE TS blindly trust the location information about a TD provided by any user without verifying neither the accuracy of the location information (*i.e.*, missing C_7) not the physical proximity of the TD (*i.e* missing C_5). This *violates the **Anonymous and proximity-aware location update** ($CD-S_1$) property* as an attacker can update and spoof the location of any TD without being physically close to them.

Location query. The location of a TD can be obtained by sending a GET request along with the emailID of the user account. This returns the location of all the TDs registered by the user.

Similar to TrackR, in the case of CUBE, an attacker can affect the TD's location information either by updating the location using the method described in Section 5.3 or *by spoofing MAC address of the TD* (*i.e.*, missing C_6). These attacks clearly *violate the **Reliable location service** ($TS-S_1$) property.*

5.4 Chipolo

In Chipolo, the TS assigns to every TD and user a unique id, called `chipoloid`, which is used by the users to communicate with the TS on behalf of the TD.

Owner registration. In Chipolo, similar to other tracking systems, a TD can be owned by first pairing with it. However, registering the ownership of a TD to the TS requires two REST calls. To register as the owner for a TD, first, the user needs to send a GET request containing the unique id assigned to the user on account registration and the Bluetooth MAC address of the TD.

If the TD is not registered TS sends a one-time secret device token. Next, the user needs to make a POST request with JSON payload containing the above secret token. On successful registration, TS sends a JSON response with a success message containing the `chipoloid` that can be used for all future communication with TS concerning the TD. However, similar to other tracking systems, the TS does not verify that the CD can communicate with the TD (*i.e.*, missing C_2). This enables an attacker to own an unregistered TD by only knowing its MAC address, which does not require physical access. This *violates the **Physical ownership** ($TD-S_2$) property*

Furthermore, unlike other tracking systems, only owners *i.e.*, paired users, can send privilege commands (using a CD) to a TDs, thus *preserving the **Privileged owners** ($TD-S_1$) property.*

Location update. Similarly to owner registration, updating the location of a TD requires two REST calls. To update the location of a TD, first, a GET request should be sent with the unique id for the user and `DEVICEID`, which is an identifier broadcasted by the TD through one of the advertised packets.

This GET request returns the `chipoloid`, which is then used in the second PUT request to the authenticated REST endpoint, with the payload containing the location information.

Similar to other systems, the TS of Chipolo blindly trusts the location information about a TD provided by any user without verifying neither the accuracy of the location information (*i.e.*, missing C_7) nor the physical proximity of the TD (*i.e.*, missing C_5). Consequently, this *violates the **Anonymous and proximity-aware location update** ($CD-S_1$) property.*

Location query. A user can obtain the location information of all the TDs by sending a GET request, which returns a JSON response containing the location information of all the TDs owned by the user corresponding to `USERID`.

As explained in Section 5.4, given the MAC address of a TD, an attacker can update the location of the TD, thereby compromising the integrity of location information. Thus *violating the **Reliable location service** ($TS-S_1$) property.*

5.5 Pebblebee

Owner registration. To register as the owner of a TD, a user needs to first pair with the TD through BLE. To register with the TS, a POST request needs to be sent with the Bluetooth MAC address of the TD. During registration, the TS does not verify that the user can communicate with the TD, consequently missing C_4. This enables an attacker to own an unregistered TD by knowing its MAC address, which does not require physical access, and, therefore, *violates the **Physical ownership** ($TD-S_2$) property.*

Similar to TrackR and CUBE, the TD in the case of Pebblebee does not keep track of the owner (*i.e.*, missing C_1) which enables an attacker to perform privileged operations, such as ringing and blink, on the TD by sending messages over BLE, thus *violating the **Privileged owners** ($TD-S_1$) property.*

Location update. In Pebblebee, location update of a TD is done by sending a POST request with a JSON payload containing the location information along with the MAC address of the TD.

Similar to other systems, the TS of Pebblebee does not verify either the accuracy of the location information (*i.e.*, missing C_7) nor the physical proximity of the TD (*i.e.*, missing C_5). Consequently, this *violates the* ***Anonymous and proximity-aware location update*** *($CD-S_1$) property.*

Location query. The location query in Pebblebee is done by sending a GET request, with the Bluetooth MAC address of the TD. Unfortunately, given the `MAC ADDRESS` of a TD, *any user* can query the location (a privileged operation) of the TD using the above URL. Furthermore, as explained in Section 5.2, from a single MAC address of a TD, it is easy to enumerate the MAC address of all TDs. Consequently, an attacker can perform location query of all TDs, which further *violates the* ***Privileged owners*** *($TD-S_1$) property.*

Given that an attacker can arbitrary update the location of any TD (Section 5.5), this affects the integrity of location information. Thus *violating the* ***Reliable location service*** *($TS-S_1$) property.*

5.6 Tile

Owner registration. Similar to the other systems, TD in Tile needs to be first paired through BLE using TApp, which will then register with TS by sending a PUT request to it's API interface at https://production.tile-api.com/api/v1/. Along with this request, a unique signature and static client uuid are added. This requires the TD to be physically present near the CD. Unlike other systems, once a Tile is added it cannot be deleted by the owner. The owner, however does have other options such as to hide TD temporarily, transfer it to another user using their email address and replace an existing Tile with a new Tile. The owner registration process of Tile is the most secure design of all the analyzed systems. In fact, our design as explained in Section 6.3 is inspired by Tile.

Location update In Tile, location updates are encrypted and updated by sending a POST request to the TS. But, only logged in users can send the location update request thereby *violating the* ***Anonymous and proximity-aware location update*** *($CD-S_1$) property.*

Location query Location query in Tile is done by sending a authenticated GET request, with requesting client uuid, and a tile request signature of the CD. Furthermore, only owner can query the location of a TD.

Finally, given that the location information is blindly trusted. An attacker can update all in-proximity TDss with random locations thereby *violating the* ***Reliable location service*** *($TS-S_1$) property.*

5.7 Summary

Table 3 summarizes the results of our analysis. We verified all our findings by implementing a Proof-of-Concept (PoC) for each violation. Interestingly, all the tacking systems satisfy two NSCs, *i.e.*, C_2 (TS able to recognize owners of a CD) and C_8 (Impossible to spoof the TS), which they achieve by using HTTPS for their communications. Our results show that, except for the **Privileged owners** ($TD-S_1$) property in Chipolo, *none of analyzed tracking systems guarantee the desired security properties.*

Responsible disclosure. All the device vendors have been contacted and notified regarding the security issues discussed in the paper. Two vendors already replied and acknowledged our findings. We have not heard back from the remaining two vendors yet.

6 SECURE TRACKING SERVICE

In this section, we present SECrow, a crowdsourced tracking system that satisfies all the desired security properties. Similar to the existing systems, the TS in SECrow requires any user to register and log in before using the system. Furthermore, the TS keeps track of the ownership association *i.e.*, the set of owners of a TD, and is also able to recognize an owner of a TD. This satisfies our NSC C_2. Furthermore, the TS uses TLS for all its communication, thereby making it impossible to spoof a TS, which satisfies NSC C_8.

6.1 Public key cryptography

One of the main building blocks of SECrow is the public key cryptography [54]. Specifically, every entity in our system, *i.e.*, TS, CD, and, TD, has a public key and private key pair. As in all public key cryptographic systems, the public key of an entity is available to the other entities through certificates. The private key is, instead, secret and only known to the corresponding entity.

Notations. We use the following notations for various cryptographic primitives. Pk_x and Pr_x indicate public and private key of entity x. $Enc(K,P)$ and $Sign(R,P)$ indicates encryption of content P with *public key* K and signature of P with R, in the case of RSA this is same as encryption of P with *private key* R. Since encryption with the private key is called a signature, we use *Sign* to indicate encryption with a private key. Also, $SEnc(S,P)$ indicates encryption with the *symmetric key* S X_I indicates a unique *identifier* (ID) for the entity X. For instance, this could be the Bluetooth MAC address for a TD, and the user *ID* of the logged-in user for a CD.

6.2 Built in capabilities

In SECrow, we assume the TDs and CDs to have certain capabilities, which allow them to provide the required security properties.

TD capabilities. The TD has its private key etched on to on-chip read-only memory, such as e-fuses [22]. Thus, to spoof a given TD, one needs access to its private key, which is impossible without physically invasive techniques [44]. The TD also has a small amount of persistent storage and the ability to generate random numbers. The persistent storage contains the *location key (i.e., L_{TD})*, which is used as the *symmetric key* to encrypt and decrypt location data of the TD (④ in Figure 4). The persistent storage also contains the public key certificates of all the owners of the TD. This enables a TD to verify whether a given CD is one of its owners and thus to satisfy the NSC C_1. Every TD has a *single* primary owner and can have multiple secondary owners. The primary owner has additional privileges and can regulate the secondary owners of the TD.

CD capabilities. All the location updates in SECrow have to be signed. Since CDs (*i.e.*, smartphones) are commonly used to communicate with the TS, CDs have to be able to provide *signed location (GPS) information*. Furthermore, the CD operating system must not be able to tamper the location information. This can be achieved by using existing techniques [42] that leverage Trusted Execution Environments (TEEs), such as ARM TrustZone [49], which is now available across almost all the smartphone brands [10]. Furthermore, CDs have the ability to create temporary but attested public-private key pairs, denoted as TPk_{CD} and TPr_{CD}, which are managed

Table 3: Verified security properties for each device, where ✓ and ✗ in each cell indicate whether the corresponding security property holds or not respectively. The NSCs (C_x) under ✗ indicates the *missing conditions (*Section 3*).*

	Security Properties of a crowdsourced tracking system			
	Tracking device (TD)		Communication device (CD)	Tracking service (TS)
Device manufacturer	Privileged owners ($TD-S_1$)	Physical ownership ($TD-S_2$)	Anonymous and proximity-aware location update ($CD-S_1$)	Reliable location service ($TS-S_1$)
TrackR	✗ (C_1)	✗ (C_3, C_4)	✗ (C_5)	✗ (C_6, C_7)
CUBE	✗ (C_1)	✗ (C_3, C_4)	✗ (C_5)	✗ (C_6, C_7)
Chipolo	✓	✗ (C_3, C_4)	✗ (C_5)	✗ (C_7)
Pebblebee	✗ (C_1)	✗ (C_3, C_4)	✗ (C_5)	✗ (C_6, C_7)
Tile	✓	✓	✗ (C_5)	✗ (C_7)

by a TEE and cannot be tampered with by the user operating system [6, 29]. As we show in Section 6.4, these keys are used to provide anonymous but signed location information. In this scheme, the CDs cannot provide fake location information, therefore satisfying one of the NSCs, *i.e.*, C_7.

6.3 Owner registration

In SECrow, adding owners to a TD requires the TD to be in pairing mode, which is only possible by physically holding down a dedicated button on the TD. This satisfies C_3 NSC (Section 3.1). Figure 3 shows the sequence diagram of the protocol used in SECrow to add the primary owner to a TD and the TS.

The first block *Adding Primary Owner* shows the messages exchanged when a CD (or user) requests to be added as the owner of the TD. When a `AddPOwner` request along with the public key, Pk_{CD}, is received by a TD, the TD sends a challenge nonce N_1 back to the CD. The CD is expected to sign the nonce N_1 to prove that it has access to the private key of the corresponding public key sent in the initial message. On successful verification of the signature by the TD, the TD adds the pubic key Pk_{CD} as the primary owner. The `AddPOwner` request is accepted by a TD *only when there is no primary owner yet* in the TD. The primary owner can add secondary owners using the `AddSOwner` command (Section 6.3).

Registering as an owner (either primary or secondary) of a TD to the TS requires three interactions, as indicated by the bottom three blocks in Figure 3. First, the CD sends a `AddOwner` request along with its ID (CD_I) and the ID of the TD (TD_I) to the TS. The TS replies with a nonce N_2 and a temporary symmetric key OT_K both encrypted with the public key of TD_I (*i.e.*, O_T). Second (`CD Owner Query`), the CD relays O_T along with its public key Pk_{CD} to the TD. The TD decrypts O_T (as it has the private key Pr_{TD}) to retrieve N_2 and the temporary *symmetric key* OT_K. Using this temporary key, OT_K, the TD sends back an encrypted message (O_{CD}) that includes the public key of the owner (Pk_{oCD}), the decrypted nonce (N_2), and, an additional nonce for freshness (N_3). Here, $Pk_{oCD} = Pk_{CD}$, if Pk_{CD} is one of the owners of the TD, else Pk_{oCD} is the public key of one of the registered owners. The use of OT_K for encryption ensures that only the TS can know the owner information of the TD, thereby protecting the privacy of the TD. Furthermore, the freshness token $N3$ prevents inference attacks. Finally, in `Commit Owner`, the CD relays O_{CD} to the TS. The TS decrypts O_{CD} as it has access to the private key of TD (Section 6.1), retrieves the Pk_{oCD}, verifies that the decrypted nonce matches N_2, and, discards N_3. If the Pk_{oCD} is the public key of CD_I then TS adds CD_I as one of the owners of the TD (or TD_I). The validation of the nonce N_2, through O_{CD}, ensures that the ownership request is approved by the TD, therefore satisfying our C_4 NSC.

The protocol described above satisfies the **Physical ownership (TD** $-$ S_2**)** property, *i.e.*, it is impossible to own a tracking device without physical access. This is because, first, adding an owner to a TD requires the TD to be in pairing mode, which is only possible by physically pressing a button on the TD. Second, to register to the TS, the CD must be able to provide *a valid* O_{CD}, *i.e.*, the public key of the owner and correct nonce (N_2), encrypted with the temporary symmetric key OT_K. The CD cannot produce a valid O_{CD} because it requires access to N_2 and OT_K, but, both are encrypted (O_T) with the public key of TD. However, generating a valid O_{CD} from O_T is possible by communicating with the TD (*i.e.* `CD Owner Query` of Figure 3). Note that, the TD generates a valid O_{CD} containing the public key of the CD if and only if the CD is one of its owners. As mentioned before, adding owners requires physical access. Instead, if the CD is not an owner of the TD, the TD still produces a valid O_T, but containing the public key of a real owner.

Primary owner commands. As mentioned in Section 6.2, the primary owner of a TD has additional privileges. Specifically, primary owners can issue the following commands: (1) `UpdateLocKey` to update the location key (*i.e.*, L_{TD}) of the TD; (2) `AddSOwner` & `RemSOwner` to add or remove secondary owners of the TD.

Each of the above commands involves a challenge nonce issued by the TD, which is encrypted with the stored public key of the primary owner. The requesting CD can decrypt the message and obtain the nonce only if it is indeed the primary owner. The decrypted nonce is used to authenticate the following requests. Figure 2 shows the flow of the `UpdateLocKey` procedure in SECrow. Here, when the command is issued by a CD (①), the TD responds (②) back with an encrypted random nonce (E_n)—encrypted with the stored public key of the primary owner (Pk_{pCD}). The CD, if it is the primary owner (*i.e.*, the CD has the corresponding private key), can decrypt E_n to retrieve N. The CD now uses N to send the data (③) of the request (R) to the TD, *i.e.*, the location key L_{TD} along with N, both encrypted with the public key of the TD. Finally, the TD, after receiving R, decrypts and verifies the nonce N. On a successful match, the TD stores the provided key, L_{TD}, as the location key. As we show in Section 6.5, the location key of a TD is required to obtain the latest location of the TD. We consider that *primary owner to be responsible for sharing the location key with secondary owners*. This provides the primary owner an access control mechanism to regulate who (secondary owners) can access the location information of the TD. Note that, by using a random nonce, our design is not vulnerable to replay attacks.

The commands `AddSOwner` and `RemSOwner(s)` follow a procedure similar to the one shown in Figure 2. However, in the request data R, the primary owner sends the public key of the CD to be added or removed instead of the location key.

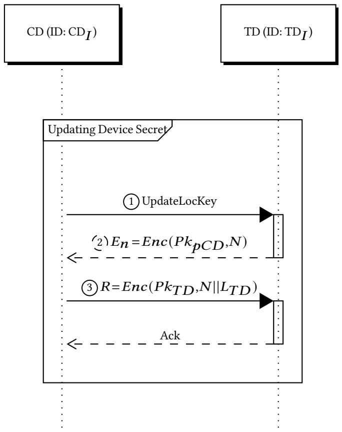

Figure 2: Updating the location key of a TD in SECrow

6.4 Location update

In SECrow, we aim to guarantee the **Anonymous and proximity-aware location update ($CD-S_1$)** property, *i.e.*, a CD should be able to anonymously update the location of a TD if and only if it is in BLE proximity of the TD. Figure 4 shows the sequence diagram of the protocol used to update the location of a TD in SECrow.

First, the CD sends a `Location update request` to the TS. This request contains the temporary and attested public-key (TPk_{CD}) of the CD and the ID of the TD whose location needs to be updated. The public-key TPk_{CD} comes with a certificate chain. The root certificate within this chain is signed using an attestation root key, which the device manufacturer injects into the device's hardware-backed keystore at the factory [30]. The TS responds back with two nonces, N_t (E_T) and N_c (E_C), which are encrypted with the public key of TD and CD respectively.

Next, as shown in the `Sign Token` block, the CD, interacting with the TEE, decrypts E_C to get N_c. The nonce N_c along with the location information (`loc`) is signed by the TEE with the temporary private key (TPr_{CD}), resulting in L. The CD forwards the encrypted nonce E_T, TPk_{CD}, and L to the TD (③). The TD verifies L using TPk_{CD} and decrypts E_T to get the nonce N_t. Next, the TD encrypts the location `loc` and a random nonce N_l using the location key to get E_L, and then signs both nonces, N_t (decrypted from E_T) and N_c (retrieved from L), to generate S_T. Both E_L and S_T are then sent to the CD (④). The random nonce, N_l, acts as a freshness token and avoid inference attacks.

Finally, as shown in `Encrypted Location Update` block, the CD forwards E_L and S_T to TS (⑤). The TS verifies S_T to check that the signatures are valid and contain expected nonces. Consequently, it updates its database with the location of the TD to be E_L.

Note that the requirement of a valid S_T satisfies our C_5 NSC. In fact, to generate a valid S_T, the CD must communicate with the TD, and, hence, it has to be in BLE proximity. The described location update protocol satisfies our **Anonymous and proximity-aware location update ($CD - S_1$)** property. First, the use of temporary keys ensures the anonymity of the CD. Second, to perform a location update, as shown in `Attested Location Update` block of Figure 4, the CD is expected to send S_T, which contains the nonces $N_t||N_c$ signed with the private key of the TD. Thus, the CD can produce a valid S_T only by communicating with the TD.

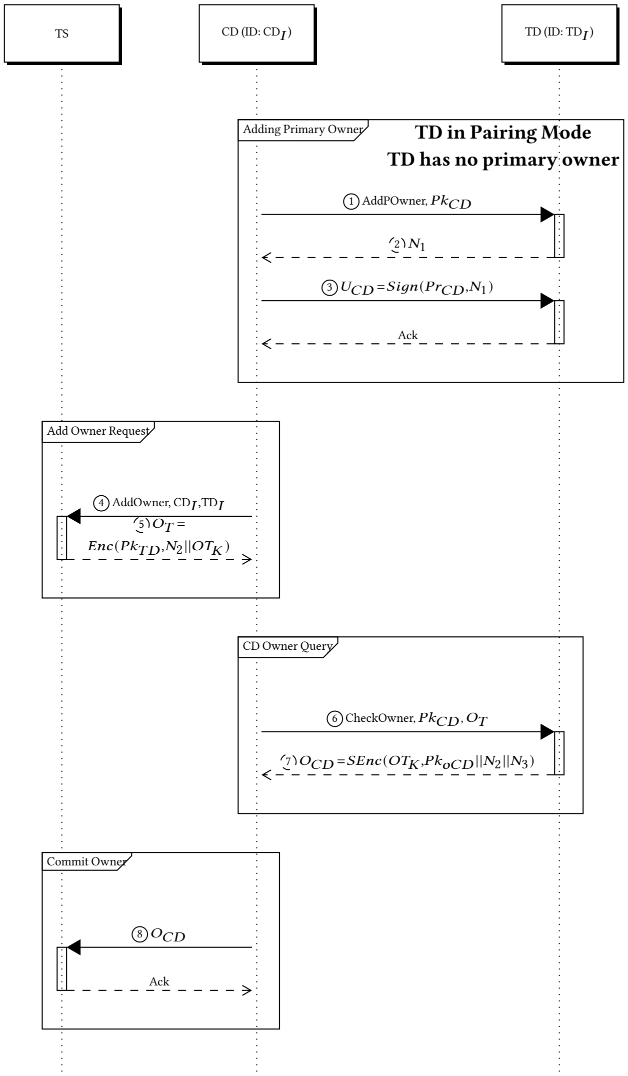

Figure 3: Secure owner registration of SECrow

Furthermore, the location information of a TD is stored on the TS in encrypted form (*i.e.*, E_L), and the decryption key is *unknown to the TS*. This design guarantees end-to-end encryption, protecting the private information (*i.e.*, location) of the TD.

6.5 Location query

We want only the owners of a TD to be able to successfully retrieve its location from the TS, as this is a privileged operation.

In SECrow, location query by a CD (or user) happens by first authenticating the CD with the TS. We use a simple challenge-response for authentication. On successful authentication, the TS then checks whether the authenticated CD is the owner of the TD. If yes, the TS responds back with the encrypted location token, *i.e.*, $E_T = Enc(Pk_{CD}, E_L || N)$, where E_L is the last-known encrypted

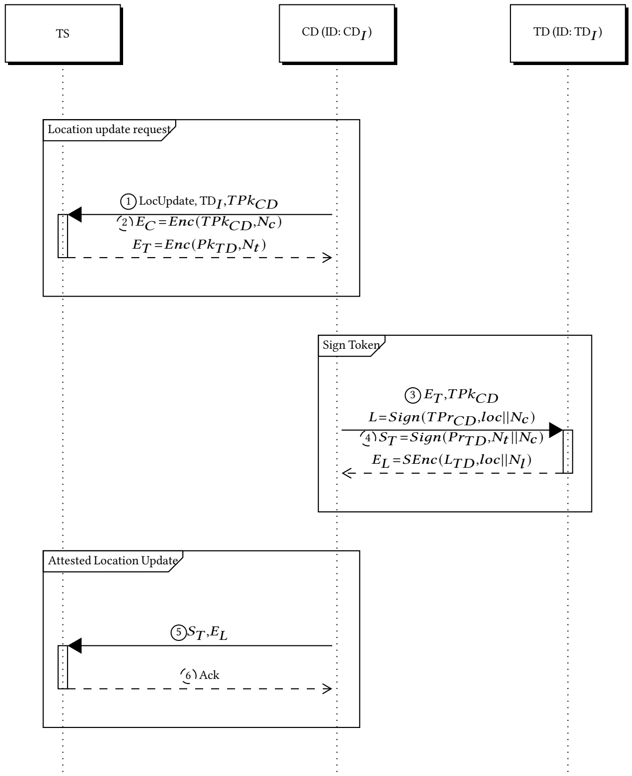
TS
CD (ID: CD_I)
TD (ID: TD_I)
Location update request
① LocUpdate, TD_I, TPk_{CD}
② $E_C = Enc(TPk_{CD}, N_c)$
$E_T = Enc(Pk_{TD}, N_t)$
Sign Token
③ E_T, TPk_{CD}
$L = Sign(TPr_{CD}, loc||N_c)$
④ $S_T = Sign(Pr_{TD}, N_t||N_c)$
$E_L = SEnc(L_{TD}, loc||N_l)$
Attested Location Update
⑤ S_T, E_L
⑥ Ack

Figure 4: Secure Location Update of SECrow

location (Section 6.4) of the requested TD, and, N is a randomly generated nonce that acts as salt to prevent against dictionary attacks to determine whether the location has changed or not.

Then, the CD decrypts E_T with its private key to obtain E_L. The CD can now use the location key (L_{TD}), previously shared with the TD, to decrypt E_L and obtain the location of the TD. As mentioned in Section 6.3, the location key is never shared with the TS. Instead, such a key has to be independently shared by the primary owner of the TD with the other secondary owners.

In summary, our location query protocol satisfies our **Privileged owners ($TD - S_1$)** property, because, to perform a location query, the CD is expected to be the registered privileged owner of the TD. Furthermore, the nonce N is randomly generated for each request, thus making it impossible for a non-owner CD to infer whether the location of a TD has changed or not.

7 EVALUATION

We implemented a preliminary prototype of SECrow using a Raspberry Pi 3B [51] with BLE as our TD. We implemented the CD and TS functionalities in Python.

In this section, we evaluate the performance overhead of SECrow in terms of execution time and energy consumption. We measure the time overhead that affects the CD, as it initiates all the operations. Whereas for energy consumption, we look at the TD as it has strict power requirements. Specifically, we measure how much additional energy is consumed by SECrow compared to the baseline system. We use a power meter [2] to measure the energy consumed by the TD (*i.e.*, Raspberry Pi 3B). We consider the unmodified TrackR system to be our baseline.

Cryptographic operations overhead. We measure the overhead of the individual cryptographic operations on the CD, TD, and TS. Table 4 shows the time required for each of the cryptographic operations along with the processors' speed. These times are very small compared to the frequency of location update and query operations. In fact, almost all the TApps limit the frequency of such operations to once in 5 minutes.

End-to-End overhead. We measure the end-to-end overhead introduced to our system, compared to our baseline (i.e., TrackR). As all the operations start and end at CD, the timings are measured at the CD. Table 5 shows the time required by each of the operations in SECrow compared to the baseline. Although the time is significantly higher compared to the baseline, the absolute numbers are still small and fall well within operational range (i.e., about 5 min). Furthermore, additional network communication (both BLE and Ethernet) in SECrow is the main contributor for the time overhead as the time for cryptographic operations is significantly low (Table 4). The power consumption of the TD for each operation is shown in Table 6. Here, we multiply the average wattage of the TD by the total amount of time of the corresponding operation. Thus, this represents an upper bound. In practice, given that most of the time is spent in the network communication between the CD and TS, the actual power consumed by the TD can be significantly lower. Furthermore, these numbers are again negligible if compared to the capacity of the modern batteries employed by the TDs.

Discussion. The performance numbers reported in our evaluation serve as an upper bound. This is because, first, our implementation was not optimized and done mainly as a proof of work to test the protocols. Second, using a Raspberry Pi 3B as a TD implied that our reported energy consumption is significantly higher than an implementation on a standard BLE device. Finally, the adoption of specialized low power cryptographic processors [46] on the TD would significantly reduce the energy consumption.

8 LIMITATIONS

SECrow relies on various communication protocols, such as TLS and BLE and implicitly assumes them to be secure. Consequently, SECrow is susceptible to any vulnerabilities [4, 9, 57] in these protocols. There are also other limitations of SECrow that are listed below:

Built in capabilities. SECrow there requires the CDs and TDs to have a few in-built capabilities (Section 6.2). One of the important capabilities expected from a TD is the ability to perform cryptographic operations. Given that TDs are battery-powered, the cryptographic operations need to be optimized to reduce power consumption. Low-power cryptographic processors are well-studied [34, 39, 55, 68]

Table 4: Execution time of the cryptographic operations (64 bytes of data with 1024-bit keys) performed on each of the entities in our implementation.

Entity	Time (ms)			
	Asymmetric Encryption	Asymmetric Decryption	Symmetric Encryption	Symmetric Decryption
TS/CD (2.6 Ghz)	3.0007e-03	2.9787e-03	0.23	0.55
TD (1.2 Ghz)	6.0101e-02	5.1594e-02	4.36	11.98

Table 5: Average (10 runs) execution time of each operation.

System	Time (seconds) Owner Registration	Primary Owner Operation	Location Update	Location Query
Baseline	0.0346	N/A	0.0386	0.0002
SECrow	4.4298	4.7319	8.6322	0.1851

and are used in various battery-powered devices [69]. These cryptographic co-processors [46] can be used to enable TDs to perform cryptographic operations without significantly affecting their battery life.

Relay attacks. Similar to contact-less systems, we assume that the ability to communicate with a TD implies proximity. However, attackers can leverage relay attacks [25] to break this assumption. In practice, a CD (proxy) could serve as a relay to a TD and could enable other CDs to communicate with the TD without being in BLE proximity. Although techniques such as distance bounding [21], secure distance measurement [41], or an out-of-band hardware token [19] could be used to prevent these attacks, the low energy requirement of the TDs makes the existing solutions impractical.

Denial-of-Service attacks. As a CD can induce a TD to perform cryptographic operations (*e.g.*, ③, ④ in Figure 4), a malicious and in-proximity CD could *drain the battery* of a TD by issuing a high number of requests. However, these attacks can be easily prevented by implementing well-known rate-limiting techniques [67].

9 RELATED WORK

The problem of effective geolocation has been well-studied. Initial works expect the devices to have the ability to communicate with a remote server [53] or the availability of trusted landmarks [31, 33]. Both of which are not feasible in our scenario.

Crowdsourced tracking. Decentralized techniques such as crowdsourcing are used to track lost objects using location-enabled smartphones [26]. SecureFind [56] provides privacy for the objects being tracked, wherein it prevents the service from accurately knowing lost devices by making the nodes (i.e., smartphones) send dummy updates to object queries. However, in SecureFind any user can determine whether a tracker device is lost, which is a privacy concern. Anonysense [17] uses differential privacy techniques such as group signatures [7] to provide anonymity for the smartphones that upload location reports. These techniques have high deployment overhead as they need an end-to-end infrastructure change, including trusted intermediate services. Techu [1] takes a first step toward exploring the security issues in crowdsourced location tracking. However, the focus in Techu is mainly on the privacy of the CDs and TDs. Moreover, their threat model less strict than ours and does not consider the issues of tracking device proximity and location spoofing, which could lead to Sybil attacks [65]. Furthermore, their solution proposes a pull model for location query, where a CD has to communicate with the other CD that performed the location update to retrieve the location. This introduces additional attack vectors to a CD, as it has to accept incoming connections to perform a location update. Recently Apple proposed FindMy service [5], which requires the TD to set up an initial key pair (ECP-224) with iCloud server. To achieve this, the TD needs to have WIFI connectivity to communicate with the iCloud server, and it requires an input mechanism for a user to enter her credentials. Unfortunately, the BLE tracking devices that we study in our paper have none of the capabilities (internet connectivity and input mechanism). Similarly, several privacy-preserving contact tracing [11, 63] techniques also expect geolocation and networking capabilities, which are unfortunately not available on BLE tracking devices. Another recent work, PrivateFind [66] performs a more detailed analysis of the security issues in Bluetooth finders. They did an excellent job in analyzing various trackers and further, propose a privacy-friendly tracking system. In this work, we broaden the analysis by considering *all the security aspects* of a crowdsourced tracking system. We identify all the necessary properties (Section 3) of a secure crowdsourced tracking system, which enable a systematic security evaluation of existing crowdsourced tracking systems (Table 3). Furthermore, our system, SECrow, has additional security guarantees such as resilience to Sybil attacks, which are missing in PrivateFind.

Table 6: Average (10 runs) energy consumed by the TD.

System	Energy consumed by the TD (Joules) Owner Registration	Primary Owner Operation	Location Update	Location Query
Baseline	0.0005	N/A	1.66E-04	0
SECrow	5.5976	6.7455	11.8824	0

Bluetooth attacks. There is a considerable amount of work that studies the security of the Bluetooth protocol [38, 52]. Recently, Antonioli et al. [4] identified a series of security flaws in the specification of Bluetooth authentication and secure connection establishment, that allows attackers to impersonate Bluetooth devices.

Geolocation and privacy. It is well understood that the sharing of location by a smartphone to any tracking service could pose privacy concerns [28, 40]. Some techniques try to protect privacy by adding noise [8, 23] or use differential privacy [3, 47, 60] techniques to hinder inferring accurate location information. We cannot use these techniques for crowdsourced tracking of BLE devices because the accuracy of the location is of real importance and is one of the primary goals of the service [27]. We also actively hinder the ability to spoof location information by using TrustZone [49]. Furthermore, work from Iasonas et al. [50] showed that it is possible to infer accurate location information even from noisy data.

10 CONCLUSIONS

In this work, we first defined and formalized a set of security properties for a generic crowdsourced location tracking system. We proposed a new design for a secure crowdsourced tracking system, SECrow. Our design successfully prevents unauthorized attackers from taking advantage of the system to obtain sensitive information or arbitrarily control tracking devices. In conclusion, we believe that SECrow will enable BLE tracking devices to utilize the benefits of the crowdsourced model while guaranteeing security and privacy.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Department of Homeland Security (DHS) (Award No. FA8750-19-2-0005) and by the Office of Naval Research (ONR) under award number N00014-17-1-2011 and N00014-20-1-2632. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the ONR or DHS. Part of this work is supported by the Dutch Research Council (NWO) in the context of the INTERSECT research program (NWA.1160.18.301).

REFERENCES

[1] Ioannis Agadakos, Jason Polakis, and Georgios Portokalidis. 2017. Techu: Open and privacy-preserving crowdsourced GPS for the masses. In *Proceedings of the 2017 Annual International Conference on Mobile Systems, Applications, and Services.* 475–487.

[2] Amazon. 2020. USB C Power meter. https://www.amazon.com/Tester-Eversame-Voltmeter-Ammeter-Braided/dp/B07MGQZHGM.

[3] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* 901–914.

[4] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. BIAS: Bluetooth Impersonation AttackS. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P).*

[5] Apple. 2020. Apple Platform Security (Page 93 Find My). https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf.

[6] Apple. 2020. Storing Keys in the Secure Enclave. https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave.

[7] Giuseppe Ateniese and Gene Tsudik. 1999. Some open issues and new directions in group signatures. In *Proceedings of the 1999 International Conference on Financial Cryptography.* Springer, 196–211.

[8] Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2014. Optimal geo-indistinguishable mechanisms for location privacy. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security.* 251–262.

[9] Matthieu Caneill and Jean-Loup Gilis. 2010. Attacks against the WiFi protocols WEP and WPA. *Journal, no. December* (2010).

[10] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA.* 18–20.

[11] Justin Chan, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, et al. 2020. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing. *arXiv preprint arXiv:2004.03544* (2020).

[12] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing.. In *Proceedings of the 2018 NDSS.*

[13] Vincent Cheval, Stéphanie Delaune, and Mark Ryan. 2014. Tests for establishing security properties. In *Proceedings of the 2014 International Symposium on Trustworthy Global Computing.* Springer, 82–96.

[14] Chipolo. 2020. Chipolo App. https://play.google.com/store/apps/details?id=chipolo.net.v3&hl=en_US.

[15] Chipolo. 2020. Chipolo Tracking. https://chipolo.net/en/.

[16] Andrea Continella, Mario Polino, Marcello Pogliani, and Stefano Zanero. 2018. There's a Hole in that Bucket! A Large-scale Analysis of Misconfigured S3 Buckets. In *Proceedings of the 2018 ACM Annual Computer Security Applications Conference (ACSAC).*

[17] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. 2008. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 2008 international conference on Mobile systems, applications, and services.* 211–224.

[18] Cube. 2020. Cube Tracking. https://cubetracker.com/.

[19] Aritra Dhar, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun. 2020. Proximitee: Hardened sgx attestation by proximity verification. In *Proceedings of the 2020 ACM Conference on Data and Application Security and Privacy.* 5–16.

[20] Tim Dierks and Eric Rescorla. 2008. The transport layer security (TLS) protocol version 1.2. (2008).

[21] Saar Drimer, Steven J Murdoch, et al. 2007. Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks.. In *Proceedings of the 2007 USENIX security symposium*, Vol. 312.

[22] Karl R Erickson, Phil C Paone, David P Paulsen, II John E Sheets, and Gregory J Uhlmann. 2017. Implementing hidden security key in eFuses. US Patent 9,570,193.

[23] Kassem Fawaz and Kang G Shin. 2014. Location privacy protection for smartphone users. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 239–250.

[24] Go Finder. 2020. Go Finder HL. https://www.gofinder.net/.

[25] Aurélien Francillon, Boris Danev, and Srdjan Capkun. 2011. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the 2011 Network and Distributed System Security Symposium (NDSS).* Eidgenössische Technische Hochschule Zürich, Department of Computer Science.

[26] Christian Frank, Philipp Bolliger, Christof Roduner, and Wolfgang Kellerer. 2007. Objects calling home: Locating objects using mobile phones. In *Proceedings of the 2007 International Conference on Pervasive Computing.* Springer, 351–368.

[27] Marcia R Friesen and Robert D McLeod. 2015. Bluetooth in intelligent transportation systems: a survey. *International Journal of Intelligent Transportation Systems Research* 13, 3 (2015), 143–153.

[28] Gabriel Ghinita. 2013. Privacy for location-based services. *Synthesis Lectures on Information Security, Privacy, & Trust* 4, 1 (2013), 1–85.

[29] Google. 2020. Hardware backed Keystore. https://source.android.com/security/keystore.

[30] Google. 2020. Verifying hardware-backed key pairs with Key Attestation. https://developer.android.com/training/articles/security-key-attestation.

[31] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. 2006. Constraint-based geolocation of internet hosts. *IEEE/ACM Transactions On Networking* 14, 6 (2006), 1219–1232.

[32] Robin Heydon and Nick Hunn. 2012. Bluetooth low energy. *CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx* (2012).

[33] Zi Hu, John Heidemann, and Yuri Pradkin. 2012. Towards geolocation of millions of IP addresses. In *Proceedings of the 2012 Internet Measurement Conference.* 123–130.

[34] Michael Hutter, Martin Feldhofer, and Johannes Wolkerstorfer. 2011. A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES Like Sardines. In *Proceedings of the 2011 Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, Claudio A. Ardagna and Jianying Zhou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 144–159.

[35] Pebblebee Inc. 2020. Pebblebee Finder. https://play.google.com/store/apps/details?id=com.pebblebee.app.hive3&hl=en_US.

[36] Tile Inc. 2020. Google Play Store: Tile. https://play.google.com/store/apps/details?id=com.thetileapp.tile&hl=en_US&gl=US.

[37] TrackR Inc. 2020. Google Play Store: TrackR - Lost Item Tracker. https://play.google.com/store/apps/details?id=com.phonehalo.itemtracker.

[38] Markus Jakobsson and Susanne Wetzel. 2001. Security weaknesses in Bluetooth. In *Proceedings of the 2001 Cryptographers' Track at the RSA Conference.* Springer, 176–191.

[39] Jens-Peter Kaps. 2006. Cryptography for Ultra-Low Power Devices. (01 2006).

[40] John Krumm. 2009. A survey of computational location privacy. *Personal and Ubiquitous Computing* 13, 6 (2009), 391–399.

[41] Patrick Leu, Mridula Singh, Marc Roeschlin, Kenneth G Paterson, and Srdjan Capkun. 2019. Message time of arrival codes: A fundamental primitive for secure distance measurement. *arXiv preprint arXiv:1911.11052* (2019).

[42] Wenhao Li, Shiyu Luo, Zhichuang Sun, Yubin Xia, Long Lu, Haibo Chen, Binyu Zang, and Haibing Guan. 2018. VBu on: Practical A estation of User-driven Operations in Mobile Apps. (2018).

[43] Cube Tracker LLC. 2020. CUBE Tracker. https://play.google.com/store/apps/details?id=com.blueskyhomesales.cube&hl=en_US.

[44] Heiko Lohrke, Shahin Tajik, Thilo Krachenfels, Christian Boit, and Jean-Pierre Seifert. 2018. Key extraction using thermal laser stimulation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 573–595.

[45] Mark Masse. 2011. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* " O'Reilly Media, Inc.".

[46] NXP. 2020. C29x: Crypto Coprocessor. https://www.nxp.com/products/processors-and-microcontrollers/legacy-mcu-mpus/crypto-coprocessors/crypto-coprocessor:C29x.

[47] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. 2010. Nearest neighbor search with strong location privacy. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 619–629.

[48] Pebblebee. 2020. Pebblebee Tracking. https://pebblebee.com/.

[49] Sandro Pinto and Nuno Santos. 2019. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–36.

[50] Iasonas Polakis, George Argyros, Theofilos Petsios, Suphannee Sivakorn, and Angelos D Keromytis. 2015. Where's wally? precise user discovery attacks in location proximity services. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security.* 817–828.

[51] Raspberry. 2020. Raspberry Pi 3B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[52] UL Muhammed Rijah, S Mosharani, S Amuthapriya, MMM Mufthas, Malikberdi Hezretov, and Dhishan Dhammearatchi. 2016. Bluetooth security analysis and solution. *International Journal of Scientific and Research Publications* 6, 4 (2016), 333–338.

[53] Thomas Ristenpart, Gabriel Maganis, Arvind Krishnamurthy, and Tadayoshi Kohno. 2008. Privacy-Preserving Location Tracking of Lost or Stolen Devices: Cryptographic Techniques and Replacing Trusted Third Parties with DHTs.. In *Proceedings of the 2008 Usenix Security Symposium.* 275–290.

[54] Arto Salomaa. 2013. *Public-key cryptography.* Springer Science & Business Media.

[55] Kirat Pal Singh and Dilip Kumar. 2012. Performance Evaluation of Low Power Encrypted MIPS Crypto Processor based on Cryptography Algorithms. (2012).

[56] Jingchao Sun, Rui Zhang, Xiaocong Jin, and Yanchao Zhang. 2015. Securefind: Secure and privacy-preserving object finding via mobile crowdsourcing. *IEEE Transactions on Wireless Communications* 15, 3 (2015), 1716–1728.

[57] Erik Tews and Martin Beck. 2009. Practical attacks against WEP and WPA. In *Proceedings of the 2009 ACM conference on Wireless network security.* 79–86.

[58] Tile. 2020. Tile Tracking. https://www.thetileapp.com/en-us/.

[59] The New York Times. 2019. Capital One Data Breach Compromises Data of Over 100 Million. https://www.nytimes.com/2019/07/29/business/capital-one-data-breach-hacked.html.

[60] Hien To, Gabriel Ghinita, and Cyrus Shahabi. 2014. A framework for protecting worker location privacy in spatial crowdsourcing. *Proceedings of the VLDB Endowment* 7, 10 (2014), 919–930.

[61] Nut Tracker. 2020. NutSpace Tracking. http://www.nutspace.com/.

[62] TrackR. 2020. TrackR Tracking. https://www.thetrackr.com/.

[63] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. 2020. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273* (2020).

[64] Secrow Full version. 2020. SECROW full version. https://drive.google.com/file/d/10tsZE_j0uCZAFd5lp6TiGZkPE6urIJPw/view?usp=sharing.

[65] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y Zhao. 2016. Defending against sybil devices in crowdsourced mapping services. In *Proceedings of the 2016 Annual International Conference on Mobile Systems, Applications, and Services*. 179–191.

[66] Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. 2020. Lost and Found: Stopping Bluetooth Finders from Leaking Private Information. *arXiv preprint arXiv:2005.08208* (2020).

[67] Cynthia Wong, Stan Bielski, Ahren Studer, and Chenxi Wang. 2005. On the effectiveness of rate limiting mechanisms. In *Proceedings of the 2005 International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*. Citeseer.

[68] Kaiyuan Yang, David Blaauw, and Dennis Sylvester. 2017. Hardware designs for security in ultra-low-power IoT systems: An overview and survey. *IEEE Micro* 37, 6 (2017), 72–89.

[69] Alexey Zhukov. 2015. Lightweight cryptography: modern development paradigms. In *Proceedings of the 2015 International Conference on Security of Information and Networks*. 7–7.

[70] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. 2019. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1296–1310.