# Blind My — An Improved Cryptographic Protocol to Prevent Stalking in Apple's Find My Network

Travis Mayberry
mayberry@usna.edu
United States Naval Academy
USA

Erik-Oliver Blass
erik-oliver.blass@airbus.com
Airbus
Germany

Ellis Fenske
fenske@usna.edu
United States Naval Academy
USA

## ABSTRACT

In 2020, Apple introduced the Find My protocol, which allows owners to crowdsource the location of their lost Apple devices even when the lost device has no active internet connection (e.g., Wi-Fi, Cellular). The Find My protocol is the basis for Apple's AirTag tracking tokens which were released later in 2021. In order to prevent malicious use of these tokens, Apple also implemented "item safety alerts" which can warn a person if they are being tracked by an AirTag without their knowledge. However, researchers have recently identified several shortcomings with these alerts that allow modified AirTags to track unsuspecting victims indefinitely without being detected. Making matters worse, while recognizing the observed malicious use of AirTags, news reports, Apple's press releases, and their intended anti-tracking improvements to the protocol do not consider the potential surreptitious use of the Find My network by custom built AirTag clones.

In this work, we present an improved Find My protocol which effectively limits the capabilities of malicious AirTags and guarantees that they can be detected while tracking. We accomplish this by adding additional cryptographic verification into the protocol, which restricts tags to only using a bounded set of keys while tracking. In order to maintain - and exceed - the privacy guarantees of the current Find My protocol, we make use of specialized *partial blind signatures*.

To demonstrate the practicality of this protocol, we implement it end-to-end using a programmable device with the same SoC (nRF52832) as in current AirTags. We also benchmark the cryptographic operations of our protocol and show that they require only modest overhead during the initial pairing procedure.

## KEYWORDS

airtag, find my, ble, tracking devices, location privacy, privacy-preserving protocols

## 1 INTRODUCTION

Physical tracking devices have been in active use for decades, but the cost, requisite communications back-haul, and maintenance associated with traditional tracking devices (e.g., GPS transponders) has limited their use to specific applications, like monitoring large equipment or performing expensive targeted surveillance. In the past few years, commercial hardware vendors have seized the opportunity presented by modern Systems on a Chip (SOCs) which are small, inexpensive, power-efficient, programmable, widely-available, and radio-enabled. These embedded systems support the production of physical tracking devices designed for the consumer market at scale. Instead of determining their own location and directly reporting it back to the owner, which would require an expensive and power-draining cellular radio, these devices rely on nearby bystanders participating in the service to crowdsource their location over close-range protocols like Wi-Fi or Bluetooth and report it back to the owner on the device's behalf.

Apple's version of this product, AirTags [6], are particularly effective because they use Apple's Find My network [7], consisting of millions of Apple devices which participate in the service by default. AirTags have only a Bluetooth radio which they use to send out "lost messages," beacons indicating to nearby iPhones and other Apple devices that they are lost. Then those "finding" devices deliver the reports to an Apple server using their internet connection. This allows AirTags to be found even when they are lost outside of Wi-Fi or cellular range, since passerby devices can record the lost message and deliver it once connectivity is reestablished.

Tracking devices like AirTags, designed to help consumers find and recover lost belongings, present a serious novel threat to the privacy of not only their users but everyone, in that they trivialize the task of location surveillance. Since their release in Spring 2021, popular media has been flooded with reports of AirTags used maliciously, for theft [17, 18, 27], stalking and domestic abuse [2, 5, 19], and to spy on well-known public figures [24]. In addition, in February 2022 Apple released a statement [1] explicitly outlining concerns about malicious AirTag use, stating they have worked with law enforcement to successfully identify and apprehend malicious actors using AirTags to perform unwanted tracking.

In an attempt to mitigate these tracking concerns, Apple has introduced Item Safety Alerts (ISAs). If an iPhone observes the same AirTag over a period of time and in separate locations, it will alert the iPhone owner that they are likely the victim of unwanted surveillance and help them identify and disable the offending AirTag. This feature is in part responsible for the media response, since in contrast to surreptitious tracking technology ISAs allow users to become aware their privacy has been violated and report it. The presence of these news reports provide some evidence that ISAs are succeeding in their intended task. Unfortunately, previous work [21] has demonstrated that these alerts can be trivially bypassed by programming inexpensive, commercially available devices to participate in the Find My network. This allows adversaries to construct one small programmable device that


*Proceedings on Privacy Enhancing Technologies 2023(1), 85–97*

https://doi.org/10.56553/popets-2023-0006

emulates multiple distinct AirTags over time. The lost messages produced by these custom devices are accepted by all Apple bystander devices (e.g., iPhones), providing malicious tracking capabilities. However, unlike genuine AirTags, these custom devices may not alert nearby monitoring systems to their persistent presence, due to the fact that they appear to be different devices at different times. Because of the privacy properties associated with the Find My network, it is difficult to detect malicious behavior of this type as the protocol is currently deployed. While Apple has indicated a list of intended implementation changes to improve anti-tracking technology for AirTags [1], none of these planned changes provide protection against malicious custom devices.

Further, other manufacturers, including Samsung, are quickly following suit by developing their own tracking solutions [26]. Apple's protocol provides many privacy guarantees in practice, but as these networks are deployed and refined, there is no clear definition for what security and privacy properties such a protocol satisfies or should satisfy.

In the following, we propose formal security definitions for a crowdsourced location tracking network like Apple's Find My network. We also introduce Blind My, a cryptographic protocol intended as a drop-in replacement to the current Find My protocol that ensures devices that send valid lost messages accepted by the network are legitimate participants, and therefore will generate ISAs. Our protocol successfully addresses the vulnerability identified by Mayberry et al. [21] and prevents third-party stealth tracking devices. We formally prove the protocol correct with respect to our security definitions, provide a fully functioning open-source implementation, and evaluate its performance on hardware comparable to an AirTag.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Prior Work

Since Apple has not publicly released technical specifications for Find My enabled devices or the Find My protocol, there is a great deal of prior work solving this problem by reverse engineering the protocol [16], extracting and analyzing the AirTag firmware [22, 25], and constructing tools to detect and analyze Find My traffic [14, 20]. Researchers have also studied ISAs [21] specifically, analyzing when they are produced and how they can be circumvented. Prior work also exists analyzing the security properties of other tracking services [30], and researchers have produced high-level informal security notions for crowdsourced location tracking networks in general [13], but these contributions analyze earlier tracking networks and do not reflect the novel security and privacy features included in the Find My network. In particular, tracking networks analyzed by prior work do not make any attempt to alert potential victims of the presence of malicious tracking devices with a feature like ISAs which are the primary focus of this work. For context, we briefly describe the Find My protocol, AirTags, Item Safety Alerts, cryptographic primitives required for our proposed protocol, and known attacks on the security properties of the Find My network.

### 2.2 Find My Protocol

The Find My protocol, introduced by Apple in 2019, allows many different devices (e.g., iPhones, iPads, MacBooks) to broadcast Bluetooth

**Table 1: Breakdown of the byte structure of a lost message in the Separated state.**

| Byte | Value | Comment |
|---|---|---|
| MAC addr | Public key | Bytes 0-5 of $PW_j$ |
| 0 | `0x12` | Payload Type |
| 1 | `0x19` | Length |
| 2 | | Battery info, see discussion |
| 3-24 | Public key | Bytes 6-27 of $PW_j$ |
| 25 | Key overflow | Bits 0-1 of byte 0 of $PW_j$ |
| 26 | Byte 5 of $P_i$ | Unknown purpose |

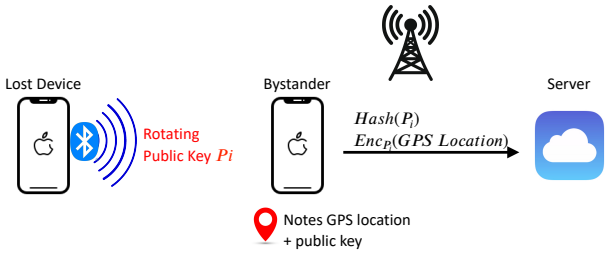


**Figure 1: Protocol execution showing the beacon broadcast from a lost device being picked up by a bystander device and a location report being encrypted and uploaded to the server.**

Low Energy (BLE) advertisement frames called *lost messages* which contain as their payload a P-224 Elliptic Curve public key. Table 1 shows the format of a lost message. Because the BLE advertisement frame is limited to a maximum of 27 application-usable bytes, the first 6 bytes of the 28-byte public key are stored as the device BLE MAC address. This means that the 6-byte MAC address used for lost messages is not generated for link-layer addressing purpose directly, as is the case with a traditional random MAC address, and instead consists of the first 6 bytes of a compressed elliptic curve point less two fixed bits required to form a valid randomized BLE MAC address.

Upon receiving a lost message, a *bystander device* that participates in the Find My protocol extracts the public key contained in the lost message ($PW_j$ in Table 1) and generates an ephemeral key with which to complete an Elliptic Curve Diffie-Hellman Key Exchange, producing a shared symmetric key. The bystander device then uses the symmetric key to encrypt a timestamp and GPS coordinates with `AES-128-GCM` into an encrypted payload called a *location report*. After some period of time and when the bystander device is able to make a connection to the Apple servers, the bystander device uploads a key-value pair to the Apple servers, where the key is the the hash of the received public key, and the value is the encrypted location report recorded by the bystander device when it received the initial lost message [16].

The Apple server stores location reports and permits queries by key of this key-value store. Based on our experience using the OpenHaystack tool [15] which facilitates building and locating custom Find My devices, the server does not appear to rate limit or prevent spurious or malicious queries, so long as the querying device has an iCloud account.

## 2.3 AirTags

Apple AirTags, introduced in 2021, are small, durable, coin-sized, inexpensive commercial products that participate in the Find My network by sending lost messages. They do not feature Wi-Fi radios or GPS technology and primarily communicate over BLE, keeping their battery usage low. Thus, their expected battery life is around one year during normal use. When purchased, AirTags must be *paired* with a nearby Apple device, at which point they attach to the iCloud account of that device's *owner*. When an AirTag is not in the proximity of an owner device (like an iPhone), it begins transmitting lost messages once per second. An AirTag changes the content of its lost messages by rotating to a new pseudo-random public key daily. Each public key is derived by application of a key derivation function (KDF) to the previous key, and all are derivable from secret, encrypted data included in the iCloud account associated with the AirTag.

We note that other devices within the Apple ecosystem produce lost messages as well: iPhones produce lost messages when they are unable to connect to Apple's servers (so they can be found by their owners in areas without a stable Internet connection), and when they are powered off [12]. AirPods, Apple's wireless headphone product, also produce lost messages when separated from their owning devices much in the way AirTags do.

## 2.4 Item Safety Alerts

With the release of AirTags, Apple also introduced an iOS feature, the Item Safety Alert (ISA) intended to notify the owner of an iPhone that they may be a victim of a malicious Find My tracking device. These alerts appear when an iPhone observes the same lost message consistently, exceeding some threshold of elapsed time and travelled distance. Researchers have attempted to experimentally determine these thresholds [21], and Apple has also publicly announced intentions to change them [22]. The ISA includes a warning that a malicious tracking device may be nearby, and instructions for how to identify and disable the device. In August 2021, researchers released a tool called AirGuard [14] which provides a similar functionality on Android devices. Later that year, Apple also released an official Android app [8] that implements similar Item Safety Alerts for Android. Apple has also just released a guide for users to verify their devices are participating in the Find My network and will receive ISAs [9] .

ISAs and AirGuard both function based on the principle that devices can detect nearby lost messages and determine that a given lost message is consistently present as the device moves and time elapses. If a device consistently receives the same lost message, it infers that there is an unknown AirTag tracking device present and alerts the user. While the ISAs built into iOS only detect Find My lost messages, the same principle can be applied to detect trackers of all types, and the AirGuard application can detect trackers from other manufacturers (such as Tile [28]) as well.

*2.4.1 Attacks on Item Safety Alerts.* Unfortunately, this approach does not detect all malicious tracking devices. Recently, Mayberry et al. [21] outlined a series of methods that defeat the ISAs technology allowing malicious tracking by devices that can participate in the Apple Find My network undetected. These devices avoid discovery and will not trigger ISAs, thereby exploiting Apple's network of millions of iPhones which dutifully act as bystander devices and report lost messages without user interaction.

The first approach taken by Mayberry et al. is to send lost messages with an invalid byte set for battery status (see Table 1). Surprisingly, the researchers were able to determine that iOS devices did not generate ISAs from lost messages constructed in this way.

Beyond this simple attack, the researchers showed that with the battery byte set correctly, trackers could quickly rotate through keys every few minutes rather than daily, which also prevented alerts from being shown. To a detection service like AirGuard or iOS, it appears as if new AirTags are present every few minutes (plausible in a densely populated environment) rather than the same tracking device remaining nearby over time. A malicious party can then query Apple's servers for all of the keys used in the rotation for a given tracking device and recover the location history as with a normal AirTag. In this way, the researchers were able to produce malicious tracking devices with a comparable cost, form factor and battery life to an AirTag that *never* produced ISAs when used to track honest users. We reproduced these attacks, verifying that at this time Apple has not patched the vulnerability.

However, we assert that even if Apple patched this original vulnerability, the Find My protocol is still deficient because malicious trackers are able to leverage the Find My network. The goal of this paper's new protocol is to introduce additional cryptographic protections that ensure each tag is given only one valid key per day and that they cannot feasibly produce any additional keys without help from the server, thus preventing malicious actors from attempting to circumvent ISAs. As a result, we can guarantee that Item Safety Alerts function properly and malicious actors cannot successfully track users without detection.

## 2.5 Partial Blind Signatures

In order to guarantee the authenticity of a user's requests to the server, but also provide anonymity to the user, our protocol makes use of blind signatures. Introduced by Chaum [11], blind signatures allow a user to interact with a *signer* to obtain a signature of their input without the signer learning what that input is. There are various applications of blind signatures, but they are principally used for rate limiting, granting valid signatures for arbitrary data while ensuring that the number of signatures issued is known and controlled by the signer.

In our application, the data being signed are the public keys that will be used by the tag. By adding this control to the protocol, we guarantee that a malicious device cannot produce more keys than it should be allowed to have, in order to circumvent tracking alerts.

It is important that we used blind signatures for this process because we must maintain the privacy guarantees that exist in the current Find My protocol. As it is now, the server cannot determine, from seeing a public key, which device that key belongs to. If we used normal digital signatures, the server would be able to correlate issued keys with a particular lost device later on, after seeing location reports for it. Blind signatures prevent this because the server does not see the actual public keys while it is signing them.

However, since the signatures will be computed blind by the server, we need an additional way to guarantee that each key is only valid for one day and that only one key is issued for each

day. For this, we employ *partial blind signatures* [3] which can additionally include some plaintext data which is visible to both the signer and client and can be confirmed by the signer during the signature process. For instance, the plaintext info in our protocol will contain the date for which each key is valid.

We use the partial blind signature scheme from Abe and Okamoto [4] which is based on the hardness of the discrete logarithm problem. We now reproduce the definitions of partial blind signatures and refer the reader to the original paper for full details of the scheme. Observe that there is a mistake in the security definition of Abe and Okamoto [4], we instead use the corrected definition from the follow-on work by Okamoto [23].

A partially blind signature scheme is composed of four algorithms:

- KeyGen is a probabilistic polynomial time (PPT) algorithm that takes a security parameter as an argument and outputs a secret and public key pair (sk, pk).
- Signer and User are interactive PPT algorithms that jointly compute a signature $\sigma$ of a message $m$, including auxiliary plaintext data info. Signer starts with (sk, pk, info). User starts with (pk, info, $m$). After interacting, User outputs $\perp$ if the protocol fails or $(m, \sigma)$ if the protocol succeeds.
- Verify is a PPT algorithm that takes as input (pk, info, $m, \sigma$) and outputs `accept` if the signature is valid for the message $m$ and auxiliary date info or `reject` if it is not.

The correctness property of a partial blind signature scheme is identical to that of a regular digital signature scheme, that a correctly issued signature will result in the verification algorithm accepting.

Definition 1 (Completeness). *Let $\lambda$ be the security parameter. If* Signer *and* User *follow the signature protocol with the same input* (pk, info) *then* User *outputs* $(m, \sigma)$ *such that* Verify(pk, info, $m, \sigma$) $\rightarrow$ `accept` *with probability* $1 - negl(\lambda)$.

The security definition of a partial blind signature scheme is similar to a blind signature, but with the additional requirement that the auxiliary data info be known to both Signer and User while also being irrevocably tied into the signature in a way that can be verified later with the public key.

There are two parts to the security definition: partial blindness, indicating that the signer does not learn the input $m$, and unforgeability, that a user not possessing the private key cannot forge a signature for any new ($m$, info).

Definition 2 (Partial blindness). *A partial blind signature scheme achieves partial blindness,* iff *for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\epsilon$ such that for sufficiently large $\lambda$*

$$|Pr[\mathsf{Exp}^{\mathsf{PartialBlind}}_{\mathcal{A},\Pi}(\lambda) = 1] - Pr[\mathsf{Exp}^{\mathsf{PartialBlind}}_{\mathcal{A},\Pi}(\lambda) = 0]| < \epsilon(\lambda).$$

That is to say, a signature scheme is partially blind if:

(1) The signer can be guaranteed that a plaintext info field is included in the signature.
(2) The signer cannot distinguish between any two signatures it has created with the same info field.

This property is extremely valuable in privacy-preserving systems because it allows the signer to have control over the rate that signatures are created as well as who those signatures are granted to, but the signatures themselves cannot be traced to the user that received them.

**Partial blindness experiment**

(1) Adversary $\mathcal{A}$ is initialized with $1^\lambda$ and outputs a public key, pk, two messages $(m_0, m_1)$ and an auxiliary data **info**.
(2) Honest challenger $C$ randomly generates bit $b \in \{0, 1\}$. Two User algorithms are initialized $U_0$ and $U_1$ such that $U_0$ gets message $m_b$ and $U_1$ gets message $m_{\bar{b}}$. Both are given info and pk.
(3) $\mathcal{A}$ engages in the signature protocol with both $U_0$ and $U_1$. $U_0$ outputs $(m_b, \sigma_b)$ and $U_1$ outputs $(m_{1-b}, \sigma_{1-b})$.
(4) $C$ gives $(m_b, \sigma_b)$ and $(m_{1-b}, \sigma_{1-b})$ to $\mathcal{A}$ in random order, to hide which user output which signature.
(5) $\mathcal{A}$ outputs $b' \in 0, 1$.

**Figure 2: Experiment** $\mathsf{Exp}^{\mathsf{PartialBlind}}_{\mathcal{A},\Pi}(\lambda)$

**Unforgeability experiment**

(1) $C$ is initialized with $1^\lambda$ and outputs a public key pk.
(2) Adversary $\mathcal{A}$ chooses auxiliary data info.
(3) $\mathcal{A}$ chooses $m$ and receives a signature $\sigma$ from $C$ such that Verify(pk, info, $m, \sigma$) = `accept`.
(4) Repeat previous step polynomially many times.
(5) $\mathcal{A}$ chooses $(m', \sigma')$ such that $m' \neq m$ for any $m$ sent in step (3).
(6) $C$ outputs 1 if Verify(pk, info, $m', \sigma'$) = `accept` and 0 otherwise.

**Figure 3: Experiment** $\mathsf{Exp}^{\mathsf{Unforgeable}}_{\mathcal{A},\Pi}(\lambda)$

Finally, we use a standard definition of unforgeability for digital signatures.

Definition 3 (Unforgeability). *A partial blind signature scheme achieves unforgeability,* iff *for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\epsilon$ such that for sufficiently large $\lambda$*

$$Pr[\mathsf{Exp}^{\mathsf{PartialBlind}}_{\mathcal{A},\Pi}(\lambda) = 1] < \epsilon(\lambda).$$

These two properties together provide a digital signature scheme which will allow us to maintain user privacy while ensuring that public keys in the Find My protocol can be rate limited and malicious trackers can be caught with the existing Item Safety Alerts.

## 3 SECURITY DEFINITIONS

In this section, we provide a definition for a crowdsourced location tracking protocol $\Pi$, list the parties that participate in $\Pi$, and give a series of formal cryptographic security definitions that reflect both the security goals the Apple Find My protocol achieves and the additional properties required to prevent tag forgery or rotation attacks such as those described in [21].

### 3.1 Parties

A crowdsourced tracking protocol $\Pi$ involves the following parties.

**Tracking device**: This is the device whose location will be tracked. It is presumed to not have ready access to the internet, but does have Bluetooth Low Energy capabilities which it utilizes to transmit lost messages ("Beacons"). Using Apple's AirTags as

a representative device, we assume limited memory (~50 kb) and CPU power.

**Owner**: The owner device is a more powerful device (laptop, smartphone, tablet), has both Bluetooth and some connection to the internet (Wi-Fi/cellular). This device has two main purposes: it acts as a proxy to the internet for the tracking device during pairing and it can query the server for locations of their lost devices. Note that the owner device can also be the tracking device, e.g. iPhones which transition to offline tracking devices when they are turned off (as of iOS 15). In case of adversarial compromise, the owner and tracking device are considered to be fully malicious (they can participate in bad faith in protocols, change messages, etc.)

**Bystander**: A device that receives lost messages from the tracking device. It can receive BLE messages and has, or will later reestablish an internet connection via a Wi-Fi or cellular network. In practice, this device is usually a smartphone. It encrypts and sends the location ("location report") of the lost tracking device to the server. This device is also fully malicious if compromised by an adversary.

**Server**: The server receives location reports from bystander devices and interacts with owners to allow them to retrieve reports for their lost tracking devices. We implicitly assume that the server maintains some kind of internal state, a database $\mathcal{D}$, where it stores, e.g., location reports. The server is considered honest-but-curious if compromised by an adversary.

**Collusion**: In our model, we allow for the bystander and server to collude arbitrarily. The tag does not collude with any parties, simply because it is the one trying to achieve privacy and has nothing to gain by colluding.

## 3.2 Tracking Protocol

Moreover, a crowdsourced tracking protocol $\Pi$ = (Pairing, Beacon, GenReport, RetrieveReports) comprises four algorithms.

(1) Pairing(serial) $\rightarrow (S_{\text{server}}, S_{\text{owner}})$: A protocol between the owner device and the server to establish any parameters or long-term secrets necessary for the tracking device to operate. While the server does not have any input to this protocol, the owner device inputs serial. The output of the protocol is $S_{\text{server}}$ for the server and $S_{\text{owner}}$ for the owner device. The protocol may also output $\perp$ if the server determines that serial is not valid.
After pairing, the owner device communicates with the tracking device to share any secrets necessary for the protocol. These devices are considered owned and controlled by one entity so their communication is not explicitly described in the security definition.

(2) Beacon$(S_{\text{owner}}, t) \rightarrow B$: The tracking device outputs a BLE advertisement beacon $B$ to broadcast based on data it has received during pairing from the owner. The tracking device also uses the current time interval $t$ as input. For simplicity, and to match the current Find My protocol, $t$ is the number of days passed since a fixed epoch (January 1, 1970). If Beacon is executed honestly, and if $t$ is outside of the range of dates that the tag supports, i.e., before the time it was paired or past the end of its service date, this function returns $\perp$. The range of dates that the tag supports is typically encoded into both $S_{\text{owner}}$ and $S_{\text{server}}$.

**Location indistinguishability experiment**

(1) Adversary $\mathcal{A}$ is initialized with $1^\lambda$ and outputs serial.
(2) Challenger $C$ chooses random bit $b \in \{0,1\}$ and runs Pairing(serial) with $\mathcal{A}$ being the semi-honest server. Consequently, $\mathcal{A}$ receives $S_{\text{server}}$, and $C$ receives $S_{\text{owner}}$.
(3) $\mathcal{A}$ outputs $t$ and two locations $\text{loc}_0$ and $\text{loc}_1$. $\mathcal{A}$ gives $t$ to $C$.
(4) $C$ computes $B \leftarrow \text{Beacon}(S_{\text{owner}}, t)$ and $R \leftarrow \text{GenReport}(B, \text{loc}_b)$. If $B = \perp$ ($t$ is invalid), output 0. Otherwise, $C$ gives $R$ to $\mathcal{A}$ who updates their internal database $\mathcal{D}$.
(5) $C$ runs RetrieveReports$(S_{\text{owner}}, S_{\text{server}}, t, \mathcal{D})$ with $\mathcal{A}$ being the semi-honest server. $C$ receives $R$ as output.
(6) $\mathcal{A}$ repeats steps 3 and 4 $poly(\lambda)$ times.
(7) $\mathcal{A}$ outputs a guess $b'$.
(8) Output 1 if $b = b'$, otherwise 0.

**Figure 4: Experiment** $\text{Exp}^{\text{Loc}}_{\mathcal{A},\Pi}(\lambda)$

(3) GenReport$(B, \text{loc}) \rightarrow R$: This is a protocol involving a tracking device and a bystander device. A bystander device receives a beacon $B$ from the tracking device and creates a location report $R$ for its current location loc.
This $R$ is typically sent to the server which updates its internal state $\mathcal{D}$.

(4) RetrieveReports$(S_{\text{owner}}, S_{\text{server}}, t, \mathcal{D}) \rightarrow R$: This is a protocol between the owner device with inputs $S_{\text{owner}}$ and $t$, and the server with inputs $S_{\text{server}}$ and $\mathcal{D}$. The owner device retrieves a set of location reports $R$ for a single tracking device for time period $t$. This protocol can fail, for example if the owner presents incorrect authentication information, in which case it returns $\perp$.

## 3.3 Definitions

Apple has achieved several important privacy properties with their current Find My protocol. In the following, we formally define these properties, to ensure that our solution also meets them, while also including new properties that support detection of malicious tracking devices.

First, we formalize the properties which are already achieved in Apple's current AirTag protocol. One of the main goals of the current protocol is that the location of tracking devices is hidden from the server. Informally, an adversarial server cannot distinguish between location reports for two different chosen locations. If this is true, it implies that the protocol effectively hides the location of the tracking device from the server.

We formalize this intuition with a standard indistinguishability definition. Consider the location indistinguishability experiment depicted in Figure 4.

Definition 4 (Location indistinguishability). *A crowdsourced tracking protocol* $\Pi$ *provides location indistinguishably* iff, *for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\epsilon$ *such that*

$$Pr[\text{Exp}^{\text{Loc}}_{\mathcal{A},\Pi}(\lambda) = 1] < 1/2 + \epsilon(\lambda).$$

In addition to hiding the location of the device, it should also be the case that the server and the bystander cannot learn any useful identifying information about the device itself when observing

**Tag indistinguishability experiment**

(1) Adversary $\mathcal{A}$ is initialized with $1^\lambda$ and outputs $\text{serial}_0$ and $\text{serial}_1$.
(2) Challenger $C$ chooses random bit $b \in \{0,1\}$ and runs Pairing($\text{serial}_0$) and Pairing($\text{serial}_1$) with $\mathcal{A}$ being the semi-honest server. $\mathcal{A}$ receives $(S_{\text{server}_0}, S_{\text{server}_1})$, and $C$ receives $(S_{\text{owner}_0}, S_{\text{owner}_1})$.
(3) $\mathcal{A}$ outputs time interval $t$ and location loc and gives it to $C$.
(4) $C$ computes $B_0 \leftarrow \text{Beacon}(S_{\text{owner}_0}, t)$, $B_1 \leftarrow \text{Beacon}(S_{\text{owner}_1}, t)$. If $B_0 = \perp$ or $B_1 = \perp$, output 0.
$C$ computes $R_0 \leftarrow \text{GenReport}(B_0, \text{loc})$, and $R_1 \leftarrow \text{GenReport}(B_1, \text{loc})$. $C$ gives $(B_0, B_1, R_0, R_1)$ to $\mathcal{A}$.
(5) $C$ runs RetrieveReports($S_{\text{owner}_0}, S_{\text{server}_0}, t, \mathcal{D}$) and RetrieveReports($S_{\text{owner}_1}, S_{\text{server}_1}, t, \mathcal{D}$) with $\mathcal{A}$ being the semi-honest server.
(6) $\mathcal{A}$ repeats steps 3 to 6 $poly(\lambda)$ times.
(7) $\mathcal{A}$ outputs a $t'$ that is not equal to any previously selected value of $t$.
(8) $C$ computes $B_0 \leftarrow \text{Beacon}(S_{\text{owner}_b}, t')$, $B_1 \leftarrow \text{Beacon}(S_{\text{owner}_{\bar{b}}}, t')$. If $B_0 = \perp$ or $B_1 = \perp$, output 0.
$C$ computes $R_0 \leftarrow \text{GenReport}(B_0, \text{loc})$, and $R_1 \leftarrow \text{GenReport}(B_1, \text{loc})$.
(9) $C$ gives $(B_0, B_1, R_0, R_1)$ to $\mathcal{A}$.
(10) $C$ runs RetrieveReports($S_{\text{owner}_b}, S_{\text{server}_0}, t, \mathcal{D}$) and RetrieveReports($S_{\text{owner}_{\bar{b}}}, S_{\text{server}_1}, t, \mathcal{D}$) with $\mathcal{A}$ being the semi-honest server.
(11) $\mathcal{A}$ outputs a guess $b'$.
(12) Output 1 if $b = b'$, otherwise 0.

**Figure 5: Experiment** $\text{Exp}^{\text{Tag}}_{\mathcal{A},\Pi}(\lambda)$

beacons. We formalize that with a similar game-based experiment, except this time varying the device identity.

Consider experiment $\text{Exp}^{\text{Tag}}_{\mathcal{A},\Pi}(\lambda)$ in Figure 5. There, an adversarial server runs the pairing protocol with two devices and then gets a beacon and location report for one of the devices, whereupon it must guess to which device these correspond (steps 7 to 11). In an initial learning phase (steps 3 to 5), the adversary also gets to receive a series of validly computed beacons and reports from both devices before being challenged to guess a final unknown one.

Definition 5 (Tag Indistinguishability). *A crowdsourced tracking protocol* $\Pi$ *provides tag indistinguishably* iff, *for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\epsilon$ *such that*

$$Pr[\text{Exp}^{\text{Tag}}_{\mathcal{A},\Pi}(\lambda) = 1] < 1/2 + \epsilon(\lambda).$$

Note that, in this definition, the adversary receives both the beacons and the location reports, meaning that the indistinguishability property applies to the bystander and the server or the pair of them colluding together. It also includes both the beacon and querying phases of the protocol, meaning that this property guarantees that the owner's location queries also cannot be linked together.

Finally, we present a new definition that aims to address the malicious tracker problem. This security definition is not met by the current AirTag protocol. In this definition, contrary to the previous ones, the adversary is the owner of a tracking device. The goal of the adversary is to create two valid beacons for a single time period and be able to retrieve the location reports corresponding to both

**Beacon unforgeability experiment**

(1) Adversary $\mathcal{A}$ is initialized with $1^\lambda$ and receives a serial from challenger $C$.
(2) $\mathcal{A}$ maliciously runs Pairing with $C$ being the server. $C$ uses serial as their input. $\mathcal{A}$ receives $S_{\text{owner}}$, and $C$ receives $S_{\text{server}}$.
(3) $\mathcal{A}$ outputs a time period $t_{\mathcal{A}}$ that is valid for the performed pairing, two locations $(\text{loc}_0, \text{loc}_1)$ such that $\text{loc}_0 \neq \text{loc}_1$, and two beacon messages $(B_0, B_1)$ such that $B_0 \neq B_1$. $\mathcal{A}$ gives $(\text{loc}_0, \text{loc}_1, B_0, B_1)$ to $C$.
(4) $C$ selects two random bits $b_0, b_1 \in \{0,1\}$ and computes $\text{GenReport}(B_0, \text{loc}_{b_0})$ and $\text{GenReport}(B_1, \text{loc}_{b_1})$ and updates internal database $\mathcal{D}$.
(5) $\mathcal{A}$ gives $t_{\mathcal{A}}$ to $C$ and maliciously runs RetrieveReports with $C$ being the server $poly(\lambda)$ number of times. $C$'s input is $S_{\text{server}}$, $t_{\mathcal{A}}$, and $\mathcal{D}$.
(6) $C$ generates a random bit $b'$ and sends it to $\mathcal{A}$
(7) $\mathcal{A}$ sends $\text{loc}'$ to $C$.
(8) Output 1 if $\text{loc}' = \text{loc}_{b_{b'}}$, otherwise 0.

**Figure 6: Experiment** $\text{Exp}^{\text{Beacon}}_{\mathcal{A},\Pi}(\lambda)$

beacons. If the adversary cannot do this, then the definition ensures that only a single beacon value can be used per time period.

Consider the experiment shown in Figure 6.

Definition 6 (Beacon Unforgeability). *A crowdsourced tracking protocol* $\Pi$ *provides beacon unforgeability* iff, *for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\epsilon$ *such that*

$$Pr[\text{Exp}^{\text{Beacon}}_{\mathcal{A},\Pi}(\lambda) = 1] < 3/4 + \epsilon(\lambda).$$

This game relies on the fact that an honest tag should only be able to have one valid beacon message per time period, which is what enables tracking devices to be detected using Item Safety Alerts. In step (3), $\mathcal{A}$ must produce two beacons $B_0$ and $B_1$ for the same time period. The challenger $C$ then generates lost reports for both beacons, randomly selecting between the locations provided by $\mathcal{A}$ to give $\mathcal{A}$ the most possible control.

If the protocol is Beacon Unforgeable, then the owner will only be able to retrieve one of these location reports and will have to guess for the other, therefore winning the game with probability 3/4 (half the time they are queried on the report they know and win with probability 1 and half the time they are queried on the one they don't know and win with probability 1/2). Any non-negligible advantage beyond 3/4 means that they were able to retrieve some information about both beacons, violating this property.

Finally, we need one further property to make sure that only one pairing operation can be performed per authorized, purchased device. We omit a full game-based definition for property, as it is straightforward.

Definition 7 (Serial Unforgeability). *A crowdsourced tracking protocol* $\Pi$ *provides serial unforgeability* iff, *for all PPT adversaries* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *can forge a* serial *such that* Pairing(serial) *does not output* $\perp$ *is less than* $\epsilon(\lambda)$.

Here we use the same notion of forgery as with digital signatures, even if $\mathcal{A}$ is given many valid serials they should not be able to forge another valid one that they have not been given. Fortunately there are many easy ways to achieve this: making the serial itself a

signature, making it a MAC, choosing randomly from a large space and having the server store a whitelist of valid serials, etc.

This property is very important: if the serial numbers are not controlled, a malicious owner can perform the pairing operation many times and retrieve secret keys for multiple tags that could be coalesced together into one device violating Beacon Unforgeability.

*3.3.1 Limitations.* The goal of Beacon Unforgeability definition is to prevent malicious tags from using more keys than they should be allowed to for a given time period, which would let them avoid being detected by Item Safety Alerts. A device participating in a protocol which has the Beacon Unforgeability property still may not be detected by the ISA system, however, due to the limitations of that system itself. If a tracking device broadcasts beacons very infrequently or limited to a small period of time during the day it might still avoid being detected.

Unfortunately, the ISA system is closed-source and we do not fully know how it works. Moreover, it is being updated regularly by Apple. The aim of our definitions and protocol is simply to guarantee that all devices are restricted to the proper key rotation schedule as legitimate AirTags are, closing the vulnerability discovered by Mayberry et al. [21] and giving the ISA system the best shot at detecting malicious trackers.

## 3.4 Existing Find My Protocol

To motivate our protocol, we briefly assess Apple's Find My protocol, as currently implemented in 2022, and demonstrate that not all of our proposed definitions are achieved. We provide proof sketches for the two security properties that Find My satisfies (Location Indistinguishability and to some extent Tag Indistinguishability). We also construct an adversary which wins the game in the Beacon Unforgeability experiment with non-negligible probability to show that Find My does not satisfy Beacon Unforgeability.

For a full detailed description of the Find My protocol, we refer to Heinrich et al. [16]. We note here for convenience a few useful facts about the protocol as it relates to our definitions. For Find My, serials are not used in the protocol past Pairing and the Pairing function simply produces a Master Beacon Key as $S_{\mathsf{owner}}$, with $S_{\mathsf{server}} = \perp$, where individual beacon keys are derived using a KDF on a rolling basis. Report retrieval requires the owner to query the server for the SHA-256 hash of the public key used to construct a beacon. We model the SHA-256 based KDF used in the Find My protocol and the hash function SHA-256 applied to each public key for retrieval as random oracles.

**Location Indistinguishability**. We reduce the Location Indistinguishability Experiment to the IND-CPA game for the symmetric encryption algorithm (in this case `AES-128-GCM`) used to encrypt location reports. Suppose an adversary $\mathcal{A}$ wins the Location Indistinguishability game with non-negligible probability. Let the adversary $\mathcal{A}'$ play the IND-CPA game for the symmetric cryptosystem above. $\mathcal{A}'$ runs $\mathcal{A}$, and each time $\mathcal{A}$ sends a pair $(\mathsf{loc}_0, \mathsf{loc}_1)$ and a $t$, $\mathcal{A}'$ picks a random P224 public key to use to calculate Beacon, and calculates $\mathsf{GenReport}(\mathsf{Beacon}(t), loc_0)$ $\mathsf{GenReport}(\mathsf{Beacon}(t), loc_1)$ to send to its challenger. $\mathcal{A}'$ gives the result from the challenger to $\mathcal{A}$ in place of $R$ in step (4), and when $\mathcal{A}$ outputs $b'$, $\mathcal{A}'$ outputs this value as its output. We note that the keys used in beacon construction are constructed via application of the KDF (i.e. a random oracle) and so are uniformly random, and therefore are identically distributed to the keys constructed by $\mathcal{A}'$, and that the key generated by $\mathcal{A}'$'s challenger is constructed via the same algorithm as the challenger in $\mathsf{Exp}^{\mathsf{Loc}}_{\mathcal{A},\Pi}(\lambda)$. This means that $R$ is identically distributed to the value produced by $\mathcal{A}'$ in place of it, and the advantage of $\mathcal{A}$, $\mathcal{A}'$ are equal and therefore both non-negligible.

**Tag Indistinguishability**. For Tag Indistinguishability, we note that except with negligible probability, the Master Beacon Keys $S_{\mathsf{Owner}_0}$ and $S_{\mathsf{Owner}_1}$ are distinct. Further, in the random oracle model, any two beacons with distinct $S_{\mathsf{Owner}}$ or $t$ are a fixed template embedded with a uniformly random P-224 public key. This means that in step (4) and step (8) of $\mathsf{Exp}^{\mathsf{Tag}}_{\mathcal{A},\Pi}(\lambda)$, $B_0$ and $B_1$ are a deterministic function of uniformly random group elements selected by a random oracle, and $R_1$, $R_0$ are directly derived from $B_0$, $B_1$. This means all of the messages $\mathcal{A}$ receives in the game are values selected independently at random that do not depend on the value of $b$, giving the required result.

However, there is a practical flaw in the way the protocol is used that makes it so that Find My currently does not achieve Tag Indistinguishability. This flaw has been described by Heinrich et al. [16]. The location server requires a user to authenticate with their iCloud credentials before retrieving location reports. The location reports are indexed by their public keys (or a hash of the public key as indicated in Figure 1), which cannot be linked to a particular tag or owner. However, the fact that the owner must be logged into their iCloud account to retrieve the reports means that the server can link particular reports to an owner, and hence distinguish between reports for different tags. This is a strange oversight given that that the key derivation part of the protocol seems to be specifically designed to prevent the server from linking public keys back to individual tags. It could be easily fixed in the future by removing iCloud authentication.

**Serial Unforgeability**. It is not known how the serial numbers are generated for the Find My protocol. Since there is no attempt to control the public keys that are used in the system, this property does not actually matter for the existing Find My protocol. As demonstrated by researchers [15, 21] it is trivial to create a malicious tag that participates in the protocol even without a serial number.

**Beacon Unforgeability**. We now present an adversary $\mathcal{A}$ that wins $\mathsf{Exp}^{\mathsf{Beacon}}_{\mathcal{A},\Pi}(\lambda)$ with probability 1 with the current Find My protocol. Thus, we demonstrate that Find My does not achieve Beacon Unforgeability and consequently also motivate our improved protocol. Adversary $\mathcal{A}$ proceeds as follows: $\mathcal{A}$ can simply ignore the Master Beacon Key, generate random keys to construct each beacon $B_0$, $B_1$, and select two random distinct locations and a random time period to complete step (3). Then $\mathcal{A}$ hashes each key and uses these hashes as its input for RetrieveReports, recovering the two locations correctly with probability 1.

This attack has been demonstrated by Mayberry et al. [21] and is possible because the current implementation of the location server does not perform any validation on the keys that are requested during report retrieval. This is Find My's main shortcoming that we address in our protocol.

# 4 BLIND MY PROTOCOL

Our Blind My protocol addresses the problem of malicious trackers by achieving the Beacon Unforgeability property defined in Section 3.3. Our improvements are confined to the Pairing and RetrieveReports portions of the protocol, with the behavior of the tag and bystander unchanged except for minor practical details to allow our implementation to work at the user level rather than the OS level (see Section 6 for details).

The main idea of Blind My is to restrict each authorized tag to possessing a single valid key per time interval (one day in Find My and our proof-of-concept implementation). Unlike the current Find My protocol, which allows the tag to create its own keys in an arbitrary manner and then retrieve them later without any validity check, Blind My has the server generate signed public keys for the tag during the pairing process. This guarantees that the tag only gets one key per time interval, and the digital signature prevents the owner from generating further keys by themselves.

This naïve solution, using regular digital signatures, would violate Tag Indistinguishability because the server granting the signed public keys would later recognize which owner they belong to when location reports are uploaded. Instead, we employ blind signatures to solve this problem. Due to the blindness property of these signatures, the server is not able to link the signed public keys that it will see later on to any single tag pairing process. It only knows that the signatures are valid, not when those signatures were created.

Yet, blind signatures again introduce a small problem: in a blind signature protocol, the owner is able to specify any plaintext that they choose to be signed by the server. A malicious user could try to get 365 keys signed that are all valid for the same day, instead of one per day, and use them to execute the rotating key attack from Mayberry et al. [21]. We fix this by using partial blind signatures, which allow the signer to include some embedded plaintext into the blind signature which is not under the control of the device owner. This way, while the owner can choose the keys however they like, they do not have control over the validity period of each key and so they will always end up with one key per time interval.

With this addition, a malicious tag can beacon any public key that it likes, but during the report retrieval portion of the protocol the server will check that each report was generated from a key that is valid for the time that it was created. Thus, if a tag tries to rotate through multiple keys in one time interval, all but one of those keys will not result in valid reports and they will be useless for tracking.

## 4.1 Limitations

The core problem in the Find My protocol that leads to the key rotation vulnerability that we are trying to address is that the owner and the tags are not required to perform any type of validation or authentication to the bystander or location server to prove that their beacon messages and public keys are formed correctly and come from an authorized device. There are many cryptographic tools that could solve this, for instance the tag could attempt to prove to the bystander in zero knowledge that the public key it was using was derived faithfully from an authorized master secret and that it was valid for the current time period. However, there are some limitations inherent in the crowdsourced tracker model that prevent the use of such "heavyweight" cryptographic tools:

**BLE Advertisements**: Bluetooth beacons are limited to 27 bytes of usable data and are non-interactive broadcasts. This severely limits the space we have to work with and makes it difficult to include any authentication information. Even as it is now, the Find My protocol struggles to fit an entire public key into this space. Apple has had to use P-224, which only gives 112 bits of security, and even resorted to some tricks (described in Section 2) to pack that into the space of an advertisement message. Furthermore, it is infeasible for a tracking tag to perform regular interactive protocols with bystander devices. There are several reasons for this:

(1) Tags and bystanders are often in motion relative to each other, sometimes very fast motion (cars passing each other on the highway). By the time a beacon is registered, the tag may already be out of range for subsequent communication.
(2) Tracking tags purposefully use only advertisement messages to prolong battery life. The Bluetooth radio is awake for a fraction of a second at a time, in order to send the advertisement, then it goes to sleep and does not listen on any channels. If the tag was required to listen constantly for connections the battery life would be drastically reduced, making AirTags far less attractive as a consumer product.
(3) Making matters worse, the energy required for computation and transmission during an interactive protocol for a tag would be orders of magnitude more than what is used during advertisement in its current form. In urban areas, it is not uncommon for beacons to generate hundreds or thousands of location reports per hour from nearby devices, so requiring any meaningful computation or signal transmission for each report produced would render AirTags nearly useless in these circumstances.

**Pairing Process**: During the one-time pairing process, the Owner's device acts as a proxy for the tag to communicate with iCloud servers. During this stage, the tag can establish shared secrets with the iCloud Servers and also the Owner device. After this step, the tag is non-interactive for the remainder of its deployment. This precludes us from using any technique that would require the tracking tag to access the internet or communicate with any party besides the bystander.

With these limitations in mind, our strategy will be to focus on the pairing and report retrieval parts of the protocol. The owner device performing these steps does not have the limitations outlined above. It is connected to the internet and can run arbitrary interactive protocols. It also has much more processing capability and access to more power.

## 4.2 Our Protocol

In the following section we present our new protocol, Blind My, designed to satisfy all the security properties in Section 3.3 as a protocol between four parties: a tracking device, server, device owner, and bystander.

*4.2.1 Setup.* We assume throughout that network communications are authenticated and end-to-end encrypted (i.e., via TLS) and we assume the following protocol infrastructure is established prior to protocol execution:

- All parties agree on an elliptic curve group with a generator, a secure Message Authentication Algorithm, and a hashing function $H$.
- The server knows a key pair $(K_S, P_S)$, where the public key $P_S$ is known to all parties. In addition, the server has a private symmetric encryption key $K_{\text{SERIAL}}$. Finally, the server also maintains a database of used serial values $D_{\text{SERIAL}}$.
- Each Tracking Device $TD_j$ has been initialized with a unique serial number and a tag constructed by a secure Message Authentication Code (MAC) algorithm applied to the serial, so that each $TD_j$ has in its internal storage:

$$\left(\text{Serial}_j, T_j = \text{MAC}_{K_{\text{SERIAL}}}(\text{Serial}_j)\right)$$

- A canonical integer representation of each calendar day is established and known to all parties via a fixed epoch. We assume all devices have synchronized clocks and may produce the integer representing the current day, which we refer to as a "daystamp" in the following, at any time. We note that this choice of 24-hour time units is arbitrary and can be adjusted freely in a protocol implementation, but we choose this time unit to mirror the currently implemented Apple protocol.
- A fixed parameter $N \in \mathbb{Z}^+$ is established and known to all parties which determines how many encryption keys are produced during the pairing algorithm. As a typical value we suggest $N = 365$, to produce one year of rotating elliptic-curve encryption keys every time the pairing algorithm is executed.

*4.2.2 Pairing.* We define our pairing protocol between the server $S$, and the owner $O$. To participate in the pairing protocol, we assume the owner $O$ has some tracking device $TD$ and has extracted its internal keys $(\text{Serial}, T)$ . The owning device then transmits the keys that result from the pairing protocol to the tracking device.

(1) The owner sends $(\text{Serial}, T)$ to $S$.
(2) $S$ verifies the tag $T$ and checks $\text{Serial} \notin D_{\text{SERIAL}}$. If either of these checks fail, $S$ aborts. Otherwise, $S$ sets $\text{Serial} \in D_{\text{SERIAL}}$ and sends public parameters for $N$ partial blind signatures to $O$.
(3) $O$ generates $N$ elliptic-curve keypairs and signing requests for the hash of the public key of each keypair using $H$, with auxiliary info for each signing request set as the daystamp for the current day and the $N-1$ following days. $O$ sends all $N$ signing requests to $S$.
(4) Server verifies the auxiliary info in the signing requests is correct (corresponds to the right range of days), and aborts if not. $S$ produces the blind signature for each and sends the $N$ blind signatures to $O$.
(5) $O$ unblinds the signatures and stores $N$ pairs of the form (KeyPair, UnblindedSignature) and transfers the $N$ public keys to the tracking device $TD$, ordered by daystamp.

*4.2.3 Beacons.*

(1) The Blind My tracking device broadcasts beacons containing the public keys it has received in order, rotating to a new one every 24 hour period according to the current datestamp. The beacon format is identical to the lost messages in the original Find My protocol, outlined in Figure 1.

*4.2.4 Bystander Devices.* Bystander devices behave identically to the original Find My protocol, we simply describe the behavior below for completeness.

(1) Upon receiving a beacon, the bystander device extracts the elliptic curve public key from the beacon and records its location data including the location, a timestamp, and a confidence value indicating the accuracy of the measurement. The bystander device compiles these values into a payload, performs ECDH with an ephemeral key and the public key from the beacon, and encrypts this payload with the shared symmetric key derived from the result of the ECDH key exchange.
(2) The bystander uploads the encrypted payload, the public ephemeral key, a timestamp, and the hash of the public beacon key to $S$, who records it in a key-value store with the key as the hash of the beacon key.

*4.2.5 Location Retrieval.* Location retrieval is similar to the original Find My protocol, but with the additional step of verifying that each hash that is requested has been blind-signed and is within a valid date range, preventing adversaries from storing many old blind-signed keys and rotating them quickly in order to avoid detection.

(1) The owner $O$ sends a set of unblinded, signed public key hashes to $S$ corresponding to the date range they are interested in retrieving, along with the corresponding info fields for each one.
(2) $S$ confirms the auxiliary information on each signature is reasonable (e.g. falls within the last 10 days) and that the signature of each hash verifies correctly.
(3) $S$ retrieves any report matching the hashes supplied and returns them to the owner, including the public key hash, the ephemeral public key, and encrypted payload, minus any reports where the timestamp does not match the correct time period from the info field (this would indicate that they key was being used outside of its intended validity period).
(4) For each report, $O$ finds the public key for the report by its hash, and uses the corresponding private key alongside the ephemeral public key included in the report to decrypt the encrypted payload and recover the timestamp, confidence, and location data associated with the report.

## 4.3 Revocation/Repairing

Our protocol can be naturally extended to allow users to perform the pairing process more than once. The server can, during pairing, record the day that keys were issued to a given serial, and give out another set of $N$ keys after $N$ days have elapsed since that date.

Further, a user may re-run the pairing process and refresh their set of keys earlier as well, i.e. if the Tracking Device ownership is transferred, or if devices are to be configured so that they refresh their keys many days before the last one expires, ensuring that devices do not run out of keys while lost. Since the remaining keys have never been used and carry no private information, this can be done by simply having the owning device present all remaining keys to the server, who stores them to a blocklist and does not permit location retrieval for those keys. Since each key is bound to a day, this blocklist can be implemented by a rotating buffer which drops keys after enough time has elapsed that the reports under

these keys would not be retrievable. After the remaining keys for a device are blocked, the server and owning device are free to initiate the pairing process again to produce a set of $N$ fresh keys.

## 5 SECURITY ANALYSIS

In this section, we will show that our protocol meets all of the security definitions we introduce in Section 3.3.

Theorem 1. *Our protocol satisfies the definition of Location Indistinguishability if the symmetric cipher used is semantically secure.*

Proof. Our protocol achieves this property in the same manner as is achieved in the existing Find My protocol, as illustrated in Section 3.4. Location information is encrypted using a semantically secure cipher with a key that is not known to the location server. This part of the protocol (beacon collection and location reporting) is not changed from the existing Find My protocol. □

For Location Indistinguishability, we cannot rely on the existing proof because our protocol introduces additional information flow between the owner and the location server. Specifically, the pairing process includes the owner receiving a series of blind signatures from the server. Therefore, we require a more thorough proof to illustrate that we still achieve Location Indistinguishability.

Fortunately, the proof is intuitive and follows directly from the definition of partial blindness. Because the server is not able to distinguish between signatures that it has granted, it cannot later distinguish between different owners/tags when the locations are queried. We show this formally by a reduction from partial blindness to Location Indistinguishability.

Theorem 2. *Our protocol satisfies the definition of Tag Indistinguishability if the signature scheme used has the partial blindness property.*

Proof. Suppose we have an adversary $\mathcal{A}$ that can distinguish with non-negligible advantage in the $\mathsf{Exp}^{\mathsf{Tag}}_{\mathcal{A},\Pi}(\lambda)$ tag. We show that we can construct an adversary $\mathcal{A}'$ that can distinguish with non-negligible advantage in the $\mathsf{Exp}^{\mathsf{PartialBlind}}_{\mathcal{A},\Pi}(\lambda)$.

When interacting with $\mathcal{A}$, $\mathcal{A}'$ acts as the challenger $C$. Importantly, this means that as part of the pairing subprotocol that $\mathcal{A}'$ can choose the keys that will be signed by $\mathcal{A}$ in any way they like. The reduction involves interleaving the two games so that the adversary $\mathcal{A}$ that breaks the tag indistinguishability game is interacting with the blind signature challenger and is forced to break that game as well. Because the signing process involves multiple messages, we have to "pause" the games and connect them together in the middle as each message of the protocol is sent.

(1) $\mathcal{A}'$ performs the pairing process with $\mathcal{A}$ (steps 1-2 of Definition 5) for all keys up to the last time interval (the keys for the "learning phase"). Here $\mathcal{A}'$ chooses random keys for each time interval and has them signed by $\mathcal{A}$
(2) For the last time interval, the keys that will be used for the "challenge", $\mathcal{A}'$ interacts with A to do the signing up to the point where it receives the signing parameters $(a_0, b_0)$ and $(a_1, b_1)$, one set of parameters for each tag, from A
(3) $\mathcal{A}'$ chooses two random elliptic curve public keys $k_0$ and $k_1$
(4) $\mathcal{A}'$ performs steps 1-4 of the partial blindness game with $m_0 = k_0$ and $m_1 = k_1$, using the $(a_0, b_0)$ as the signing parameters for $U_0$ and $(a_1, b_1)$ as the parameters for $U_1$. $\mathcal{A}'$ receives $e_0$ and $e_1$ as signature requests (second message of the blind signature protocol) from $U_0$ and $U_1$
(5) $\mathcal{A}'$ sends $e_0$ and $e_1$ to $\mathcal{A}$ to continue the pairing process, receiving $(r_0, c_0, s_0, d_0)$ and $(r_1, c_1, s_1, d_1)$ from A
(6) $\mathcal{A}'$ sends $(r_0, c_0, s_0, d_0)$ and $(r_1, c_1, s_1, d_1)$ to $U_0$ and $U_1$ respectively to finish the signing process for the partial blindness game, receiving $sig_0$ and $sig_1$ as output from $U_0$ and $U_1$
(7) $\mathcal{A}'$ continues to interact with $\mathcal{A}$ in the game from Definition 5, creating beacons and reports for the learning phase
(8) In the challenge phase, $\mathcal{A}'$ creates beacons and reports from $k_0$ and $k_1$ and sends them to $\mathcal{A}$
(9) $\mathcal{A}$ outputs guess b, $\mathcal{A}'$ outputs the same guess $b$ for its game

The goal of $\mathcal{A}'$ is to figure out whether $U_0$ or $U_1$ signed $m_0$ and $m_1$, and since $m_0$ and $m_1$ are set to be the keys used in our tag indistinguishability game this is equivalent to figuring out if the beacons sent in step 7 correspond to tag 0 or tag 1.

The reduction requires only one instantiation of $\mathcal{A}$ and so is tight. If $\mathcal{A}$ has a non-negligible advantage in distinguishing between tags then $\mathcal{A}'$ will have a non-negligible advantage in distinguishing between users in the partial blindness game. □

Finally, we also have to prove that our protocol meets the new definition that Find My does not, Beacon Unforgeability. Again, this follows directly from the blind signature's unforgeability property.

**Table 2: Comparison of the existing Find My protocol and our improved Blind My protocol. Note that Tag Indistinguishability can be achieved with the Find My protocol but it currently does not due to an implementation flaw [16].**

| Security Property | Find My | Blind My |
|---|---|---|
| Location Indistinguishability | ✓ | ✓ |
| Tag Indistinguishability | ✓† | ✓ |
| Beacon Unforgeability | ✗ | ✓ |
| Serial Unforgeability | - | ✓ |

Theorem 3. *Our protocol satisfies the definition of Beacon Unforgeability if the signature scheme used is unforgeable.*

Proof. Again, we can show this via reduction from an adversary $\mathcal{A}$ that wins the game $\mathsf{Exp}^{\mathsf{Beacon}}_{\mathcal{A},\Pi}(\lambda)$ with probability higher than $3/4 + \epsilon(\lambda)$ to an adversary $\mathcal{A}'$ that wins the game $\mathsf{Exp}^{\mathsf{Unforgeable}}_{\mathcal{A},\Pi}(\lambda)$ with non-negligible probability.

Because the server, in step (2) of the $\mathsf{Exp}^{\mathsf{Beacon}}_{\mathcal{A},\Pi}(\lambda)$ game, issues only one signed key per time interval, the reduction is very simple. Adversary $\mathcal{A}'$ runs $\mathcal{A}$, acting as the challenger $C$, and uses their challenger $C'$ to create the blind signatures.

Since the server in our protocol will only respond with location reports if the client can produce a valid signature for the key being requested, and there was only one signature granted for the time period $t_{\mathcal{A}}$, only one of the locations can be retrieved honestly. Then $\mathcal{A}$ can only gain information about the second location with non-negligible probability, which it must have in order to win the game with sufficiently high probability, by forging another signed key.

**Figure 7: A Puck.js, AirTag, and quarter next to each other for reference. The Puck.js uses the same SoC (nRF52832) as the AirTag.**

Therefore, in step (5) when $\mathcal{A}$ executes RetrieveReports, with non-negligible probability $\mathcal{A}'$ will receive a forged signature from $\mathcal{A}$. $\mathcal{A}'$ then uses this signature to win its own game.

$\mathcal{A}$ requires only one instantiation of $\mathcal{A}'$ so this reduction is tight as well. □

Finally, it is easy to show that our protocol meets Serial Unforgeability as well.

Theorem 4. *Our protocol satisfies the definition of Serial Unforgeability if the MAC used is unforgeable.*

Proof. Serial numbers are assigned as a MAC computed with a secret key known only to the server. If the MAC is unforgeable then the serial will also be unforgeable. □

We summarize the results of our security analysis as applied both to the original Find My protocol and our proposed Blind My protocol in Table 2.

*Covert Channel Defense.* Researchers have demonstrated another flaw with Apple's Find My protocol that allows for a low-bandwidth covert channel to be formed [10**?** ]. A sending device can choose to broadcast beacons with chosen public keys that encode information. A receiving device queries the location server for a set of possible keys and the presence of location reports indicates that the sending device chose those keys, corresponding to some encoded message. This allows one-way communication via nearby anonymous proxies and with the sender only possessing a Bluetooth radio.

Although it is not the main goal, our protocol also mitigates these covert channel attacks. Since only one key is valid per time period, the data transmission rate across this channel is constrained significantly in our protocol, to one bit per day for every valid AirTag the adversary controls.

## 6 EVALUATION

We have implemented our protocol in order to show that the addition of extra cryptographic protections incurs a low overhead to all parties. The server and owner were implemented in Python. We have released this portion of the code as an open-source project to allow for verification of our efficiency results[1].

The tracking tag was implemented with the Puck.js platform, depicted in Figure 7. The Puck.js can broadcast arbitrary BLE advertisements, which allows us to create our own protocol that uses BLE. Blind My requires that the tracking tag stores a large number of public keys, to create new beacons each day. Fortunately these keys are small (28 bytes each), and the Puck has 512 KiB of flash memory which is more than enough.

We have also implemented the bystander as an iOS app, to most closely match the existing ecosystem. It could similarly be implemented as an Android app, or on any other platform that can receive BLE advertisement messages.

We have formatted our beacon messages as close to those in the current Find My protocol as possible, to demonstrate that our protocol could be adopted without changing hardware. Theoretically, our protocol could be completely compatible with the existing Find My advertisement specification. However, security restrictions for non-privileged apps on iOS cause any advertisement with the Apple company ID `0x004c` to be filtered out by iOS before it can reach a normal app such as our bystander app.

Furthermore, we had the additional limitation that we could not see the source MAC address of the lost messages. As depicted in Figure 1, part of the public key is stored in the random bytes of the MAC address. Again, for security reasons, non-privileged apps on iOS cannot see the MAC address of nearby Bluetooth devices.

To address both of these problems, we have slightly changed the format of the beacon message. We removed all of the extra information (battery status, payload type, reserved byte) and used all 27 bytes of the advertisement message to store the key. However, the key is actually 28 bytes long. Therefore, the final byte of the key was stored in the least-significant byte of the company ID. In theory, is is not allowed by the Bluetooth specification, meaning our proof-of-concept could not be widely deployed as an unprivileged app, but none of these problems exist at the OS level. Apple (or another mobile OS developer) could implement our protocol, as they do now with Find My, while maintaining backward compatibility with existing protocols and not violating any specifications.

With those modifications, we are able to demonstrate an end-to-end working implementation of our protocol that can track our Puck.js tags effectively. Figure 8 includes an image of our iOS tracking app showing beacon messages that have been logged and have their locations reported.

Our Blind My protocol, compared to the existing AirTag protocol, is unchanged between the AirTag and the bystander and between the bystander and the (location) server. Therefore, we focus our evaluation on the pairing algorithm, as that is where the majority of our improvements lie.

The signatures are done as a batch, meaning the pairing process only requires two network round trips, making it robust even with high network latency. The amount of communication required to pair 365 keys is approximately 100 KiB.

Although partially blind signatures are well-established in the literature, there have not been any published benchmarks evaluating their real-world efficiency. To show that our algorithm is reasonable in practice, we have benchmarked our implementation for a varying number of keys. Figure **??** shows the results. For our target use case of one pairing operation per year, e.g., 365 keys, our Python implementation takes approximately 4.5 seconds to complete the pairing, with the tests performed on a consumer laptop with an Intel Core i5-7200U processor.

[1] https://www.dropbox.com/s/txnsj8vu7ktu050/blindmy.zip?dl=0

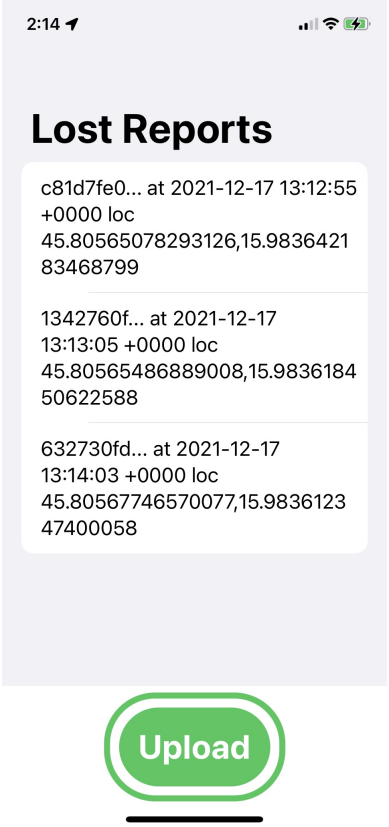


**Figure 8: Our bystander iOS App. It lists beacon keys that it has seen, with coordinates and timestamps. The upload button encrypts them and uploads to the location server.**

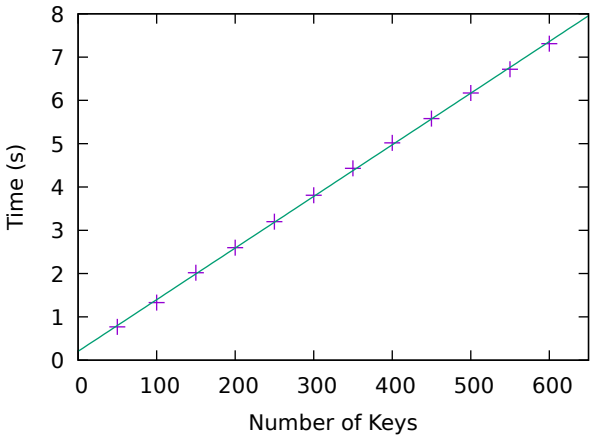


**Figure 9: Runtime of the pairing algorithm between the tag owner and the location server.**

The report retrieval algorithm is also similar to the current Find My protocol. The only additional burden is on the server to check that (1) the signature is correct and the daystamp of the report matches the daystamp of the key that is being used to retrieve it, and (2) that the key is not on the blacklist of revoked keys (if using the repairing procedure described above). These are very low overhead operations, especially considering no more than 10 keys should be retrieved at a time, based on the validity window in the current Find My protocol.

*Limitations.* Our protocol guarantees that each authorized device can only have one valid key per time period. However, a fundamental limitation of a crowdsourced tracking protocol is that a malicious adversary can always purchase multiple tags and use them together to avoid detection.

For example, an adversary could buy 30 AirTags, pair each of them with the server, and then extract the keys from them. This would give 30 valid keys for each time interval and the adversary could then load those keys onto a custom device and rotate between them as in Mayberry et al. [21]. We note that this attack is impossible to defend against with any protocol changes as long as the tag devices can have their memory dumped. The only long-term solution would be to use tamper-resistant tag hardware, which would increase their cost substantially.

Fortunately, previous work has determined that a minimum of 25 rotating keys per day is necessary to evade detection by Apple's Item Safety Alerts. This appears to be a software limitation and could be tuned at some point in the future to require more keys, but even 25 AirTags would cost over $750. For that price, a rational adversary would produce a much more sophisticated tracker that would not rely on crowdsourcing locations. So, we believe Blind My effectively stops attacks where low cost devices are added to the Find My network and serve as malicious trackers.

## 7 CONCLUSION

In this work we present the first formal definitions for a privacy-preserving crowdsourced tracking protocol that is secure against malicious tracking devices. We start by creating definitions that codify the existing privacy guarantees present in Apple's Find My protocol, then add an additional property called Beacon Unforgeability which addresses a significant flaw that allows malicious tracking devices to be used covertly to track unsuspecting third parties.

We proceed to describe an improved protocol, using partial blind signatures, that achieves Beacon Unforgeability as well. We implement this protocol and demonstrate that it is a practical replacement for the Find My protocol which could be implemented on existing hardware with minimal overhead.

## ACKNOWLEDGMENTS

We would like to acknowledge the work of our students, Noah Hollar, Devin Allen and Noah Garciagalan, in helping to test our protocol and run experiments. We would also like to thank Jeremy Martin, Chris Hoffmeister and Christine Fossaceca from MITRE, Dane Brown from USNA and Florian Kerschbaum from the University of Waterloo for help brainstorming and feedback on the paper.

# REFERENCES


[1] 2022. *An update on AirTag and unwanted tracking* (Feb 2022). https://nr.apple.com/d2I9N827r9

[2] 2022. *Atlanta Woman Can't Find Apple AirTag on Her Car That's Tracking Her Every Move.* https://www.insideedition.com/atlanta-woman-cant-find-apple-airtag-on-her-car-thats-tracking-her-every-move-72629

[3] Masayuki Abe and Eiichiro Fujisaki. 1996. How to date blind signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 244–251.

[4] Masayuki Abe and Tatsuaki Okamoto. 2000. Provably secure partially blind signatures. In *Annual International Cryptology Conference*. Springer, 271–286.

[5] Michael Allison. 2021. *Apple's AirTags are too good at tracking — that's a problem.* https://www.digitaltrends.com/mobile/airtags-stalking-problem/

[6] Apple. 2021. *AirTag - Technical Specifications.* https://support.apple.com/kb/SP840?locale=en_US

[7] Apple. 2021. *Find My security.* https://support.apple.com/guide/security/find-my-security-sec6cbc80fd0/1/web/1

[8] Apple. 2021. Tracker detect. https://play.google.com/store/apps/details?id=com.apple.trackerdetect

[9] Apple. 2022. *What to do if you get an alert that an AirTag, Find My network accessory, or set of AirPods is with you.* https://support.apple.com/en-us/HT212227

[10] Fabian Bräunlein. 2021. Send my: Arbitrary data transmission via Apple's find my network. https://positive.security/blog/send-my

[11] David Chaum. 1983. Blind signatures for untraceable payments. In *Advances in cryptology*. Springer, 199–203.

[12] Jiska Classen, Alexander Heinrich, Robert Reith, and Matthias Hollick. 2022. Evil Never Sleeps: When Wireless Malware Stays On After Turning Off iPhones. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 146–156.

[13] Chinmay Garg, Aravind Machiry, Andrea Continella, Christopher Kruegel, and Giovanni Vigna. 2021. Toward a secure crowdsourced location tracking system. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 311–322.

[14] Alexander Heinrich, Niklas Bittner, and Matthias Hollick. 2022. AirGuard-Protecting Android Users from Stalking Attacks by Apple Find My Devices. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 26–38.

[15] Alexander Heinrich, Milan Stute, and Matthias Hollick. 2021. OpenHaystack: a framework for tracking personal bluetooth devices via Apple's massive find my network. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 374–376.

[16] Alexander Heinrich, Milan Stute, Tim Kornhuber, and Matthias Hollick. 2021. Who can find my devices? security and privacy of apple's crowd-sourced bluetooth location tracking system. *arXiv preprint arXiv:2103.02282* (2021).

[17] Ryam Mac Kashmir Hill. 2021. *Are Apple AirTags Being Used to Track People and Steal Cars?* https://www.nytimes.com/2021/12/30/technology/apple-airtags-tracking-stalking.html

[18] David Ingram. 2021. *A tracking device made by Apple is showing up in suspected crimes.* https://www.nbcnews.com/news/apple-airtag-showing-up-crimes-rcna9416

[19] Albert Khoury. 2021. *A woman found an AirTag hidden under her car – Here's how to spot them.* https://www.komando.com/security-privacy/a-woman-found-an-airtag-hidden-under-her-car-heres-how-to-spot-them/820282/

[20] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik Rye, Brandon Sipes, and Sam Teplov. 2019. Handoff All Your Privacy–A Review of Apple's Bluetooth Low Energy Continuity Protocol. *Proceedings on Privacy Enhancing Technologies 2019* 4 (2019), 34–53.

[21] Travis Mayberry, Ellis Fenske, Dane Brown, Jeremy Martin, Christine Fossaceca, Erik C Rye, Sam Teplov, and Lucas Foppe. 2021. Who Tracks the Trackers? Circumventing Apple's Anti-Tracking Alerts in the Find My Network. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 181–186.

[22] Chance Miller. 2021. . https://9to5mac.com/2021/06/23/airtag-firmware-update-rolling-out/

[23] Tatsuaki Okamoto. 2006. Efficient blind and partially blind signatures without random oracles. In *Theory of Cryptography Conference*. Springer, 80–99.

[24] Elizabeth Prann. 2022. *A woman found an AirTag hidden under her car – Here's how to spot them.* https://www.cnn.com/videos/tech/2022/01/17/women-report-being-tracked-apple-airtags-mxp-prann-vpx.hln

[25] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. 2022. *AirTag of the Clones: Shenanigans with Liberated Item Finders.* IEEE.

[26] Samsung. 2022. *Use The Samsung Galaxy SmartTag and SmartTag+.* https://www.samsung.com/us/support/answer/ANS00088244/

[27] Chris Smith. 2021. *Thieves can use AirTags to track and steal your car from your driveway.* https://bgr.com/tech/thieves-can-use-airtags-to-track-and-steal-your-car-from-your-driveway/

[28] Tile. 2020. *How Does the Tile Network Work.* https://www.thetileapp.com/en-eu/blog/what-is-tile-network-community-find-lost-stolen-far-away

[29] ]tonettomy Leonardo Tonetto, Andrea Carrara, Aaron Yi Ding, and Jörg Ott. [n. d.]. Where Is My Tag? Unveiling Alternative Uses of the Apple FindMy Service. ([n. d.]).

[30] Mira Weller, Jiska Classen, Fabian Ullrich, Denis Waßmann, and Erik Tews. 2020. Lost and found: stopping bluetooth finders from leaking private information. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 184–194.