Katholieke
Universiteit
Leuven

Master of Engineering Science
Artificial Intelligence

PRIVACY IMPACT ASSESSMENT

Macless-Haystack

Lili De Bruyn (r0848166)
Annemerel Bobbaers (r0897155)
Magnus De Maerteleire (r0889456)
Jeff Jambe (r0902205)

Academic year 2025–2026

Contents

1 Introduction

This report presents a Privacy Impact Assessment (PIA) for Macless-Haystack [**?**], an open-source project that enables users to access Apple's Find My network without requiring Apple hardware. As location tracking systems become increasingly prevalent, understanding the privacy implications of such technologies is crucial for both developers and end users.

The assessment examines how Macless-Haystack collects, processes, and stores location data, identifies potential privacy risks, and proposes technical solutions to mitigate these concerns. We analyze the system from multiple perspectives: the tracker devices themselves, the Macless-Haystack application and API, and Apple's Find My infrastructure that the project relies upon.

This report is structured in three main parts. First, we provide a detailed description of the application's functionality, stakeholders, and data collection practices. Second, we conduct a privacy impact assessment that includes a threat model, identification of privacy issues, and analysis of specific threats. Finally, we present recommendations for addressing the identified privacy concerns, with particular focus on technical implementations that can be tested and verified.

2 Application Description

2.1 Functionality of the Application

The project we are analyzing is called Macless-Haystack, an open-source implementation to use Apple's 'Find My' network without needing access to an Apple device. It builds upon the OpenHaystack [**?**] project, which reverse engineered the Apple 'Find My' protocol that enables tracking of Bluetooth devices without the need of an internet connection.

At a high level, the Find My trackers broadcast identifiers, which can be detected by nearby Apple devices. These apple devices then anonymously upload the location of the detected signal to iCloud. The owner can later retrieve the location of their device by requesting reports containing the trackers identifier.

Figure **??** below visually represents the four main stages of the application:

1. **Pair through initial setup**: The process begins with the owner pairing their device, this means generating a unique private-public key pair. These keys are programmed onto supported Bluetooth devices.
2. **Broadcast**: After setup, the tracker begins broadcasting Bluetooth advertisements that are based on its public key. Apple devices nearby act as a finder device and can pick up on these advertisements.
3. **Upload encrypted location reports**: When a finder device detects an advertisement, it generates a report containing its current GPS location. This location is then encrypted using the advertisement key broadcasted by the device, making it inaccessible to everyone except the owner. The encrypted report is uploaded to Apple's servers, with no way for Apple to decipher the actual location.
4. **Download and decrypt location reports**: Any device authenticated with the owner's Apple ID can request location reports linked to the tracker's keys. Using the matching advertisement key, the owner can decrypt these reports and see each location on a map.

Macless-Haystack gets rid of the need for any Apple hardware. All you need is an Apple ID with two-factor authentication via SMS and either a linux device or a supported bluetooth enabled microcontroller.

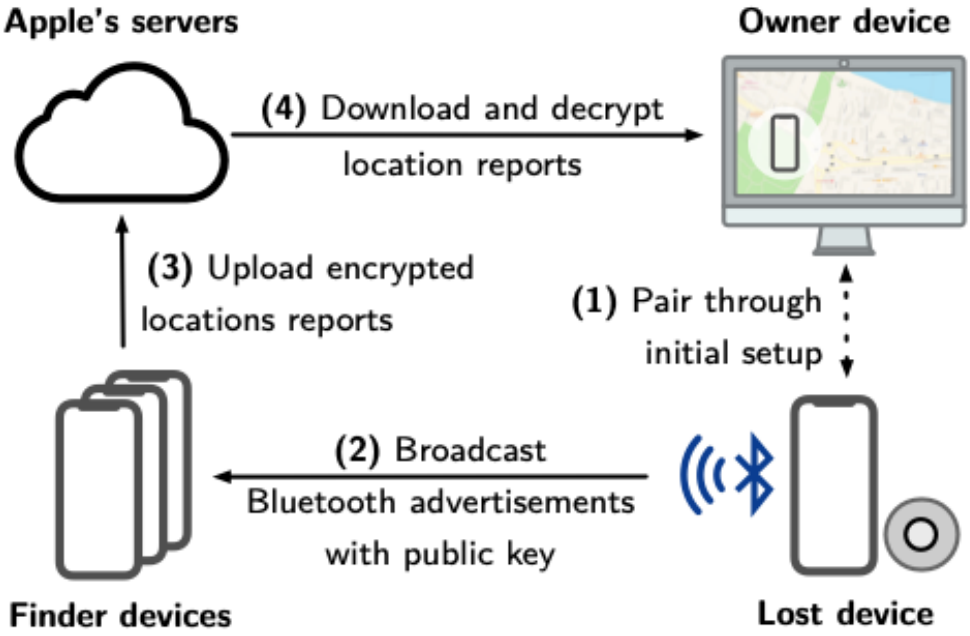

Figure 1: High-level overview of the Macless-Haystack workflow

2.2 The Stakeholders

The primary stakeholders are the end users who want to track their own Bluetooth devices via Apple's Find My network using Macless-Haystack's web browser. They provide an Apple ID, password, and SMS-based two-factor authentication once when setting up the Macless-Haystack endpoint container, and then interact with the system via a browser to import device key JSON files, view locations, filter by date, and export data. End users are responsible for generating and managing tracker keys and deciding which devices are associated with their Apple ID; they are the primary beneficiaries of the system's tracking functionality.

The Macless-Haystack endpoint operator is the party running the backend container that logs into Apple's Find My service and periodically fetches encrypted location reports, typically on a server with Docker installed. In many cases this is the same party as the end user.

The Anisette server operator runs the separate Anisette-compatible service that Macless-Haystack connects to in order to emulate a real Apple device during login and communication with Apple. This role can again coincide with the endpoint operator in a self-hosted setup, but when a public or third-party Anisette instance is used, that operator becomes a distinct stakeholder who may see sensitive authentication-related traffic and must therefore be explicitly trusted.

Apple is an indirect but essential stakeholder, as its Find My and iCloud infrastructure receives the encrypted location reports from nearby Apple devices and serves them to the Macless-Haystack endpoint authenticated with an Apple ID.

Add stakeholders diagram

2.3 Types of Data Collected and Purposes

2.3.1 Data Collection in the Tracker

The tracker itself does not collect any data. It operates solely using the public key that is provided to it.

2.3.2 Data Collection in Apple's System

Because Apple's implementation is closed-source, we can only make informed assumptions about what data is collected. For completeness, we assume that Apple collects all data that is technically available to it. Crowd-sourced Apple devices collect only three pieces of information when they detect a tracker:

- the advertisement key,
- the device's location at the moment of detection,
- and the Bluetooth signal strength at that time.

According to a blog by a Digital Forensics researcher at cellebrite[**?**], this data is normally deleted quickly, but users can delay deletion by placing their device in airplane mode. This data is placed on the apple servers, and used for people to retrieve the location of their tracker reference.

Apple stores the advertisement key together with its corresponding location report on their servers. From this, Apple can infer how many trackers are in use and estimate the general region in which they are being used, based on the uploader's IP address. Apple also knows the Apple ID associated with each device that uploads these reports. This means, at least in theory, that Apple could link a user to their tracker because the user's own Apple device will likely often be the one uploading the tracker's location reports.

Fetching location reports requires authentication. As a result, Apple can see which Apple account is hosting a macless-haystack endpoint and which trackers are used through that endpoint. This could allow Apple to terminate the Apple ID associated with the endpoint. The requirement for Apple ID authentication is likely in place to prevent abuse such as users uploading spam, overwhelming Apple's servers, or attempting to download large volumes of location reports that aren't their own.

2.3.3 Data Collected by the Macless-Haystack Application

The Macless-Haystack frontend stores very little data. The single-page browser interface keeps the private keys of imported trackers in secure local storage. It also stores the username and password used to authenticate with the macless-haystack endpoint; however, these credentials are stored in plain text. During the use of the application, the current and past location of all trackers is stored in cache.

2.3.4 Data Collected by the Macless-Haystack API

The API stores only minimal data, but it logs a significant amount of sensitive information. This includes every request made by the application, which means that if the logs are reviewed, the advertisement keys of all Macless-Haystack trackers can be seen.

Table 1: Summary of Data Collection, Storage, and Retention

Data Item	**Source**	**Storage Location**	**Retention Period**	**Potential Privacy Impact**
Apple ID / Credentials	endpoint operator	JSON in API Backend	Indefinite	Full account compromise; access to other Apple services.
public Unauthorized access to (historical) location data.	private Key(s)	End User	Browser Local Storage & JSON in filesystem & tracker firmware / flash storage	Indefinite
Advertisement Keys	Tracker / Finder Device	Apple Servers, API Logs	Apple: < 24h; Logs: Persistent	Linkability; device/owner movement profiling by nearby observers.
Location Reports	Finder Device	Apple Servers	Short-term ($\sim$24h)	Exposure of real-time/historical physical locations and routines.

3 Privacy Impact Assessment

3.1 Threat Model

The system's privacy properties depend on several trust assumptions. Users must trust Apple's infrastructure to provide end-to-end encryption (though Apple has metadata visibility), the endpoint operator to protect stored credentials and logs, and any third-party Anisette server operator to handle authentication securely. The primary trust boundaries exist at: (1) Bluetooth broadcast between tracker and nearby devices, (2) network communication between frontend and backend (currently vulnerable when HTTP is used), (3) authenticated connections to Apple's servers (which reveal query metadata), and (4) filesystem access on the server (where plaintext credentials are stored). These boundaries represent the main points where information can be compromised, as detailed in the threat analysis below.

3.2 LINDDUN Privacy Threat Analysis

The following table presents a systematic analysis of privacy threats using the LINDDUN framework, which categorizes threats across privacy dimensions: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance. Many compliance-related threats stem from the project's reverse-engineering of Apple's proprietary protocol and operation outside Apple's official ecosystem, which cannot be fully resolved without changing the project's purpose.

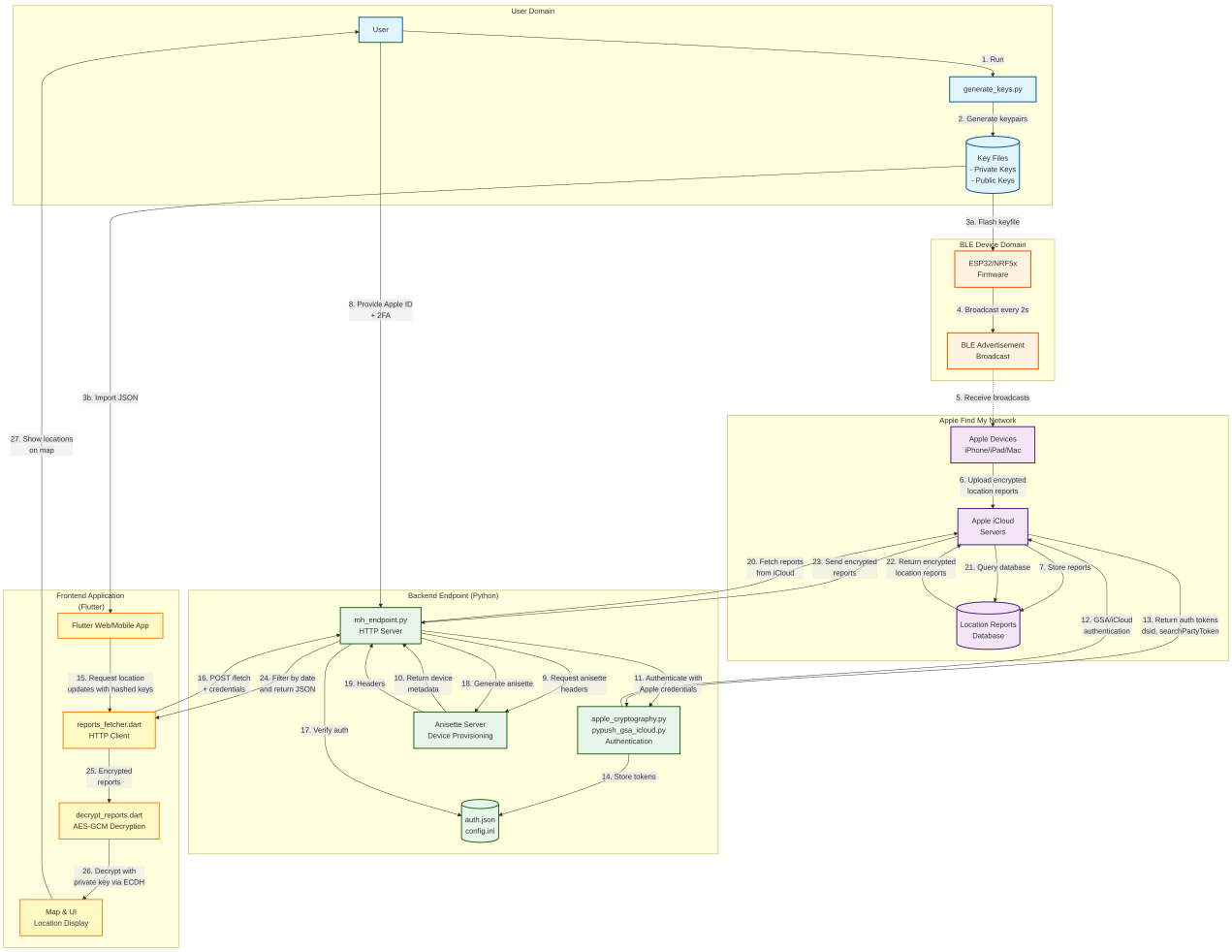

Figure 2: Macless-Haystack Architecture and Data Flow

Code	Threat Category	Description	Affected Interaction(s)	Severity	Mitigation
NC.3 / II.1	Non-Compliance	Apple ID and password can be stored in plain text in config.ini (optional but supported). Exposes credentials to anyone with filesystem access (to the macless-haystack server). Similar for auth.json	Interaction 8, 14, 13	Critical	Remove plain text password storage; require interactive login only

Code	Threat Category	Description	Affected Interaction(s)	Severity	Mitigation
DD.4.2 / NC.2	Data Disclosure / Non-Compliance	Storage of private and public keys in plain text. Keys should be stored using platform-specific secure storage (Apple Keychain, Android Keystore, GNOME Keyring, Windows Credential Manager).	Interaction 2, 3b	High	Implement secure key storage using platform APIs (e.g., flutter_secure_storage)
L.1.1 / I.2.2.b	Linkability / Information Disclosure	Static advertisement keys allow tracking of device presence. The system broadcasts the same public key continuously, enabling anyone to determine if the tracked device (and by extension, its owner) is nearby.	Interaction 4	High	Implement key rotation mechanism (firmware currently rotates every 30 min for multi-key setups)
NC.3.c	Non-Compliance	Weak authentication: no authentication required by default, or single username/password shared among all users of an endpoint instance.	Interaction 15, 16, 17, 24	High	Implement per-user authentication; support OAuth or token-based auth
U.1.2	Unawareness	Users are not informed that sharing their private key with others enables complete location tracking of their device. Lacks clear privacy warnings.	Interaction 2	Medium	Add explicit privacy warnings in documentation and UI when exporting keys
NC.1.2.a	Regulatory Non-Compliance	Violation of Apple's EULA by creating custom AirTag-like devices using Apple's Find My network infrastructure.	Interaction 3a, 4	Medium	Add legal disclaimer; users assume responsibility

Code	Threat Category	Description	Affected Interaction(s)	Severity	Mitigation
D.3	Detectability	Static advertisement keys allow querying Apple servers to determine if a tracker is still active by checking for new location reports, even without decryption.	Interaction 4, 20	Medium	Key rotation partially mitigates this; inherent to Find My protocol
NR.1.1.a	Non-Repudiation	Apple ID is attached to all location report requests, allowing Apple to identify and track Macless-Haystack server hosters and their query patterns.	Interaction 8, 11, 12, 13, 20	Medium	Inherent to Apple's authentication; no mitigation without protocol changes
NC.1.2.b	Regulatory Non-Compliance	Reverse engineering and accessing iCloud authentication from non-Apple devices violates Apple's EULA	Interaction 11, 12, 13	Medium	Legal disclaimer; users assume legal responsibility
L.2.2.1	Linkability	Sending advertisement keys to potentially public endpoint links user's IP address with their tracked devices.	Interaction 15, 16	Medium	Only use trusted, self-hosted endpoints; implement end-to-end encryption for API
NC.3.b	Non-Compliance	HTTP access is supported (non-mandatory HTTPS), exposing advertisement keys during transmission. Man-in-the-middle attackers can capture key.	Interaction 15, 16, 24	Medium	Enforce HTTPS mandatory; disable HTTP in production
I.1.2	Information Disclosure	Apple servers receive query metadata (timestamps, query frequency, advertisement key hashes) revealing usage patterns.	Interaction 20, 21	Medium	Inherent to protocol; rate limiting and query batching can reduce exposure

3.3 GDPR Compliance Considerations

Macless-Haystack processes location data, which constitutes personal data under GDPR when relating to identifiable EU individuals. For personal use, GDPR obligations are minimal under the household exemption (Article 2(2)(c)). However, organizational deployments require full GDPR compliance, including appropriate security measures (Article 32), data subject rights support (Articles 15-22), and privacy by design (Article 25). The current implementation fails to meet several GDPR requirements: plaintext credential storage and

optional HTTP violate security obligations (Article 32), while the lack of data export/deletion mechanisms prevents users from exercising their rights (Articles 15-22).

3.4 Main Identified Privacy Threats

3.4.1 Advertisement Keys in Macless Haystack

In the current setup, each tracker continuously broadcasts a beacon containing the same static public key so that nearby Apple devices can hear it and create encrypted location reports. Any Apple device that receives such a beacon, takes its own location and encrypts that location to send it to Apple's servers, where only the trackers matching private key can decrypt it. Using a static public key introduces a serious linkability privacy problem. Because the same identifier is broadcasted in every packet, any nearby device with a simple BLE sniffer can record this constant over time and across different places, and then correlate all sightings as belonging to the same physical tracker. This means that anyone could build a detailed movement profile of the tagged object or person just by re-observing the same static key at different locations, even though they cannot decrypt the underlying Find My location reports. The root cause of the privacy threat is therefore not that the encryption is weak, it is that the identifier used as input to the encrypted reporting process is long-lived and public. The encryption protects the content of each report from being read, but it does nothing to stop third parties from recognizing that "this is the same beacon as yesterday" and inferring sensitive patterns (home, workplace, daily routines) purely from repeated radio observations tied to an unchanging public key.

- **unit test:** A dedicated testing environment and test suite were developed to evaluate and validate privacy-related behaviors in the tracker firmware, which had not previously undergone testing. To determine if static keys are being utilized, the `run_openhaystack` function is called repeatedly while a shim replaces the Bluetooth advertising routine, enabling direct inspection of the broadcast payloads. Static codes are detected if multiple calls to `run_openhaystack` trigger the BLE controller with identical payloads.[1]

3.4.2 Plain Text Credentials

The user's Apple ID credentials are stored in plaintext on the Macless server inside the auth.json file. This means that anyone with file access to the server can read the user's credentials. This represents a serious security vulnerability that must be addressed immediately, especially when the server and client are managed by different individuals. Additionally, the frontend website is not protected by HTTPS, and communication between the frontend and the Macless Haystack server is transmitted without encryption. This creates a severe security risk if a user enters their credentials for the macless haystack server into the frontend, as the information would be sent over the network unencrypted. While the current implementation does not require users to enter their credentials, since they are provided during server setup, an uninformed user might still attempt to do so. In scenarios where the server is configured by a third party, entering credentials through the website may even be necessary for the system to function, further increasing the risk.

Add unit test for plain text credentials

4 Recommendations

4.1 Handeling the GDPR and EULA violations

4.2 Mitigating Linkability with Rolling Codes

Static advertisement keys allow anyone to track a device by listening for its constant signal. To fix this, we recommend using rolling codes, which means the key changes periodically (for example, every 15 minutes). This prevents strangers from linking different signals to the same device, but the owner can still find it because they have the secret key to calculate the list of all past and future IDs.
To prevent this we can use the same method as Apple's official airtags [**?**]. This uses Elliptic Curve Cryptography on the NIST P-224 curve. It starts with a master key pair and a 32-bit symmetric key. New keys are

[1] The firmware supports the use of multiple static keys; however, it was tested in a configuration where only a single key was loaded.

created using the ANSI X.963 key derivation function (KDF) with these formulas:

$$SK_i = \text{KDF}(SK_{i-1}, \text{“update”}, 32) \tag{1}$$
$$(u_i, v_i) = \text{KDF}(SK_i, \text{“diversify”}, 72) \tag{2}$$
$$d_i = (d_0 * u_i) + v_i \tag{3}$$
$$p_i = d_i * G \tag{4}$$

This requires a small firmware update. The following code snippet shows the main rolling key logic implemented on the tracker:

```c
/**
 * @brief Main rolling key logic
 * 1. Rotate Symmetric Key
 * 2. Generate u, v scalars
 * 3. Calculate new Private Key
 * 4. Derive new Public Key
 */
void roll_key_and_update_state() {
    uint8_t next_sym_key[32];
    uint8_t diversify_material[72];
    uint8_t u_bytes[36];
    uint8_t v_bytes[36];
    uint8_t rolling_priv_bytes[28];
    ESP_LOGI(LOG_TAG, "Rolling keys...");

    // 1. Update Symmetric Key: SK_new = KDF(SK_old, "update", 32)
    ansi_x963_kdf(current_symmetric_key, 32, "update", 32, next_sym_key);

    // global state update
    memcpy(current_symmetric_key, next_sym_key, 32);

    // 2. Derive u, v: KDF(SK_new, "diversify", 72)
    ansi_x963_kdf(next_sym_key, 32, "diversify", 72, diversify_material);
    memcpy(u_bytes, diversify_material, 36);
    memcpy(v_bytes, diversify_material + 36, 36);

    // 3. Math: d_i = (d_0 * u + v) mod n
    mbedtls_mpi d_0, u, v, n, d_i, temp;
    mbedtls_mpi_init(&d_0); mbedtls_mpi_init(&u); mbedtls_mpi_init(&v);
    mbedtls_mpi_init(&n);   mbedtls_mpi_init(&d_i); mbedtls_mpi_init(&temp);

    mbedtls_mpi_read_string(&n, 16, P224_ORDER_HEX);

    mbedtls_mpi_read_binary(&d_0, master_private_key, 28);
    mbedtls_mpi_read_binary(&u, u_bytes, 36);
    mbedtls_mpi_read_binary(&v, v_bytes, 36);

    // temp = d_0 * u
    mbedtls_mpi_mul_mpi(&temp, &d_0, &u);
    // d_i = temp + v
    mbedtls_mpi_add_mpi(&d_i, &temp, &v);
    // d_i = d_i mod n
    mbedtls_mpi_mod_mpi(&d_i, &d_i, &n);

    // Export result
    mbedtls_mpi_write_binary(&d_i, rolling_priv_bytes, 28);
    ESP_LOGI(LOG_TAG, "New Private Key:");
    {
        char key_str[28 * 2 + 1];
        for (int i = 0; i < 28; i++) {
            sprintf(&key_str[i * 2], "%02x", rolling_priv_bytes[i]);
        }
        key_str[28 * 2] = '\0';
        ESP_LOGI(LOG_TAG, "%s", key_str);
    }

    // Cleanup MPI
    mbedtls_mpi_free(&d_0); mbedtls_mpi_free(&u); mbedtls_mpi_free(&v);
    mbedtls_mpi_free(&n);   mbedtls_mpi_free(&d_i); mbedtls_mpi_free(&temp);

    // 4. Derive Public Key
    derive_public_key_bytes(rolling_priv_bytes, 28, current_public_key);
}
```

Listing 1: Rolling Key Implementation

The tracker needs to store the master keys and a function to calculate the new ones. The tracker needs to store the master keys and a function to calculate the new ones. This is easy to do on chips like the ESP32 or nrf5x. We tested this approach and confirmed its functionality. Additionally, the tracker should also save the current

key to memory so it doesn't lose its place if the battery dies. The major downside of this approach is that if the key-rolling interval is set to a short value, the tracker can evade Apple's built-in anti-stalking features. As a result, unlike regular AirTags, people may not receive alerts when an unknown tracker is following them allowing [**?**].

4.2.1 Client-Side Challenges

Updating the user application is challenging because custom trackers are less reliable than official AirTags, and users may not open the application for weeks or even months at a time.
As a result, when a user returns to the application after a long period of inactivity, we cannot assume that the current AirTag code is simply the last known code incremented by the elapsed time. The tracker may have been powered off for a significant portion of that period, making the actual code state unpredictable.
This uncertainty forces us to search backward from the estimated current code to the last known code. However, performing this search naively risks triggering Apple server rate limits or bans. Additionally, there is currently no built-in mechanism to resynchronize the tracker with the application.
Resynchronization could be implemented through Bluetooth or NFC, or by integrating an RTC module into the tracker. An RTC would provide accurate timekeeping, allowing the application to reliably calculate code updates even after extended powered off periods[2]

5 Conclusion

Add conclusion summarizing key findings and recommendations

[2]RTC modules typically include a built-in backup battery to maintain timekeeping when the host device is unpowered. Battery life can range from several months to over ten years, depending on usage. In our use case, the RTC would only draw backup power when the tracker is turned off, resulting in a multi-year battery lifespan.