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.

The diagram above 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.

Add technical architecture diagram

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.

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.

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.

Expand on data collection details and purposes

Add technical architecture diagram showing data flows

3 Privacy Impact Assessment

3.1 Threat Model

Start with threat model covering:

- Information at risk

- Potential adversary

- Trust assumptions (including trust boundaries)

- Trust boundaries diagram

3.2 Privacy Issues

Figure out the privacy issues that an adversary can create given the technical diagram of part 1. Feel free to prioritize issues, for the report to be interesting to read and technically sound.

List and describe privacy issues identified

Prioritize issues

Create privacy issues table

3.3 Main Identified Privacy Threats

3.3.1 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 treats the advertisement key as a valid identifier, takes its own location and encrypts that location to send it to Apple's servers, where only the 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.

3.3.2 Unit Test for Rolling Codes

A dedicated testing environment and accompanying test suite were created to evaluate and validate privacy-related behaviors in the tracker firmware, which had not previously undergone testing. To determine whether rolling keys were implemented, the run_openhaystack function is invoked repeatedly while a shim replaces the Bluetooth advertising routine, enabling direct inspection of the broadcast payloads. Rolling codes are deemed absent if each invocation of run_openhaystack consistently triggers the BLE controller with identical payload.

Test has to fail instead of succeed when rolling codes are not implemented

Add test code snippet

3.3.3 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

Add more privacy threats as identified

3.4 Test Cases

Privacy issues should also be captured in test cases, which the relevant testing framework of the selected project can run automatically. At least 2 issues should have a test that, upon completion, inform the development organization if they are fixed or not. We recommend you to capture a privacy issue that is specific enough to locate in the code base by looking at the system diagram rather than by exploring the code base to find a privacy issue. If there is no implemented recommendations, the tests should fail.

Add test cases for identified privacy issues

Ensure at least 2 issues have automated tests

Add screenshot of test suite running

4 Recommendations

This section discusses how to address the privacy issues from Part 2 with technological or methodological modifications. These modifications may be justified with legal and technical requirements, including a sensible cost-benefit analysis that balances them and provide nuance to the recommendations. The technical architecture of Part 1 should be adapted to reflect the new technologies and methodological considerations introduced, and any prioritization from part 2 should be taken into account. We expect a high degree of originality in this part.

We encourage you to implement those recommendations that address the privacy test cases you created for Part 2. This is a requirement for a top grade.

4.1 Rolling Codes

Rolling codes (or rotating keys) address the linkability problem by changing the public key that is broadcasted over time, while still allowing the real owner to decrypt all the corresponding location reports. Instead of a single static key pair, the tracker starts from a master secret and uses it to derive a sequence of public-private key pairs, for example one per 15-minute time window. At any given moment, the beacon only broadcasts the current public key, and nearby Apple devices encrypt their location reports to that key exactly as before, then upload the location report to the server. To outsiders passively listening on Bluetooth, these public keys appear as unrelated random values: the identifier at 10:00 and the identifier at 10:15 look completely different, and there is no way for them to tell that both belong to the same physical tracker. In contrast, the owner, who knows the master secret, can recompute the full schedule of rolling keys for all past time slots, query the server for reports associated with each of those keys, and decrypt them with the corresponding private keys. In other words, rolling codes preserve end-to-end encryption for the owner but break the simple "always the same identifier" signal that enables third-party tracking with static keys. Adopting rotating keys would therefore significantly reduce the privacy risk: it becomes much harder for random observers, or for large sensor deployments, to follow a specific tracker over long periods, because the broadcast identifier they see keeps changing in a way they cannot link back to one device.

Add technical justification for rolling codes

Add cost-benefit analysis

Add updated technical architecture diagram showing rolling codes

Add implementation details if implemented

4.2 Secure Credential Storage

Recommend secure storage for Apple ID credentials

Recommend encryption at rest for auth.json

Add technical justification and cost-benefit analysis

4.3 HTTPS Enforcement

Recommend mandatory HTTPS for frontend

Recommend disabling HTTP in production

Add technical justification and cost-benefit analysis

Add updated architecture diagram

4.4 Additional Recommendations

Add more recommendations based on identified privacy issues

For each recommendation:

- Technical justification
- Legal requirements (if applicable)
- Cost-benefit analysis
- Updated architecture diagrams
- Implementation status

5 Conclusion

Add conclusion summarizing key findings and recommendations

References

Add bibliography with citations