Openhaystack-Integration für den Tracking-Server

https://gemini.google.com/app/0df78d9f46fca84a

User prompt: Openhaystack soll Positionsdaten liefern. Diese Daten sollen auf möglichst einfachem und zuverlässigem Weg an meinen Trackingserver bei einem Hostinganbieter übertragen werden. Der Trackingserver ist von mir in Python geschrieben. Bitte skizziere Lösungsvorschläge und benenne die Voraussetzungen, damit dies funktionieren kann. Wäre es beispielsweise möglich, den Empfänger der Find-me Positionsdaten als Bibliothek in den Tracking server zu integrieren oder muss diese Software in einer eigenen Umgebung laufen und kommuniziert mit dem Tracking Server über eine eigen Schnittstelle (Web API)?

Response: Die kurze Antwort vorab: **Ja, theoretisch kannst du den Empfänger und Entschlüsseler als Python-Bibliothek direkt in deinen Tracking-Server integrieren**, da es Open-Source-Python-Bibliotheken (wie z. B. `FindMy.py` oder `openhaystack-python` mit `AirTagCrypto.py`) gibt, welche die Kommunikation mit Apples Find-My-Diensten und die Entschlüsselung der Standorte übernehmen.

Aus architektonischer und betrieblicher Sicht ist es jedoch **meistens ratsam (und oft stabiler), die Software in einem separaten Daemon/Skript laufen zu lassen**, das die Daten per Web-API an deinen Tracking-Server übermittelt. Der Grund dafür liegt primär in der Authentifizierung bei Apple und den Eigenheiten von Hosting-Servern.

Architektur-Vergleich: Bibliothek vs. Getrennter Dienst (Web API)

Kriterium	Direkte Integration (Als Python-Lib)	Getrennter Dienst / Daemon + Web API
Architektur	Monolithisch (alles in einer App/Codebase).	Microservice-ähnlich (Fetching-Skript $\rightarrow$ Web-API $\rightarrow$ Tracking-Server).
Wartbarkeit	Einfacher, da keine interne Schnittstelle nötig ist.	Klar getrennte Verantwortlichkeiten (Entkopplung).
Fehlertoleranz	**Schlecht:** Schlägt die Apple-Authentifizierung fehl (z. B. durch Rate-Limits, Captchas oder IP-Blockaden), leidet ggf. der ganze Server oder blockiert Threads.	**Gut:** Hängt das Abruf-Skript, läuft dein Hauptserver für andere Tracker (z. B. GPS-Tracker via MQTT/HTTP) völlig unbeeinträchtigt weiter.
Apple-Auth / IP-Probleme	Hosting-IPs (AWS, Hetzner, DigitalOcean) werden von Apple oft scharf überwacht oder blockiert.	Lässt sich lokal (z. B. auf einem Raspberry Pi daheim) oder auf einem flexibleren Server einfacher kapseln.

Lösungsvorschläge für die Umsetzung

Lösung 1: Der separate Abruf-Daemon (Empfohlen)

Ein kleines Python-Skript (oder ein Container) läuft in regelmäßigen Abständen (z. B. alle 15–30 Minuten, da das Find-My-Netzwerk ohnehin keine Echtzeit-Sekundentakte liefert):

1. Das Skript loggt sich über eine Python-Bibliothek (wie `FindMy.py`) bei Apple ein, holt die verschlüsselten Standortberichte für deine Openhaystack-Keys ab und entschlüsselt sie mit deinem Private Key.
2. Es schickt die neuen GPS-Koordinaten per HTTP-POST (Webhook / REST-API) an deinen bestehenden Python-Tracking-Server (z. B. an ein FastAPI- oder Flask-Endpoint).
3. Dein Hauptserver verarbeitet und speichert die Daten in der Datenbank.

Vorteil: Wenn Apple die Verbindung blockiert oder eine 2FA-Erneuerung fällig ist, bricht nicht dein gesamtes Tracking-Backend zusammen.

Lösung 2: Direkte Integration als Hintergrund-Task (z. B. Celery oder Asyncio)

Du bindest die Logik direkt in deinen bestehenden Python-Server (z. B. Django, FastAPI, Flask) ein. Ein interner Scheduler (wie `APScheduler` oder ein Celery-Worker) ruft im Hintergrund die Apple-API auf und schreibt die Daten direkt in die Datenbank. *Vorteil:* Du sparst dir die Implementierung einer internen Web-API-Schnittstelle zwischen den Diensten. *Nachteil:* Debugging und Fehlerbehandlung bei Apple-Sperren sind schwieriger, und blocking I/O oder Authentifizierungsfehler können den Webserver belasten.

Notwendige Voraussetzungen

Damit das Ganze – egal mit welcher Variante – zuverlässig funktioniert, müssen folgende Voraussetzungen erfüllt sein:

1. **Ein dedizierter Apple-Account (Burner-Account):** Du solltest für den Abruf **niemals** deinen privaten Haupt-Apple-Account verwenden. Erstelle einen separaten Apple-Account, da Apple bei automatisierten Abfragen oder verdächtigen Logins (siehe nächster Punkt) den Account temporär sperren oder eine Zwei-Faktor-Authentifizierung (2FA) erzwingen kann.

2. **Beachtung der Infrastruktur / IP-Adressen (Hosting-Anbieter-Problem):** Apple sichert seine Find-My-Server stark ab. Wenn du versuchst, dich von der IP-Adresse eines typischen Cloud-Hosters (Hetzner, AWS etc.) bei Apple anzumelden, schlagen oft Sicherheitsmechanismen an (Anti-Bot, erzwungene 2FA, Captchas). *Lösung:* Viele Entwickler lassen das Abruf-Skript daher auf einem Gerät mit einer normalen Heim-IP (z. B. einem Raspberry Pi zu Hause) laufen, welches die Daten dann an den Tracking-Server beim Hosting-Anbieter weiterleitet. Alternativ benötigt man Mechanismen zur Anisette-Generierung (Geräte-Fingerabdrücke), was den Betrieb auf reinen Cloud-Servern komplizierter macht.

3. **Schlüsselmanagement (Private/Public Keys):** Dein Server muss Zugriff auf den **Private Key** des jeweiligen Openhaystack-Tags haben, um die von fremden iPhones hochgeladenen, verschlüsselten Blobs entschlüsseln zu können.

4. **Sinnvolle Abfrageintervalle (Polling):** Das Apple Find My Netzwerk ist **kein Echtzeit-GPS-Tracker**. Ein Standort aktualisiert sich nur dann, wenn ein fremdes Apple-Gerät in die Bluetooth-Reichweite deines Tags kommt und dieses Signal an Apple hochlädt. Es macht daher keinen Sinn, sekündlich abzufragen. Ein Intervall von **15 bis 60 Minuten** ist üblich und schont zudem deinen Apple-Account vor Rate-Limits oder Sperren.

Sources

1. FindMy.py: The Python Tool That Locates Anything Apple
2. GitHub - seemoo-lab/openhaystack: Build your own 'AirTags' today

---

User prompt: Beschreibe bitte im Detail die notwendigen Schritte zur Inbetriebnahme eines Tags beginnend mit der Installation von Openhaystack auf einem Raspberry pi.

Response: Um einen eigenen Openhaystack-Tag (z. B. auf Basis eines ESP32, eines BBC micro:bit oder eines modifizierten BLE-Geräts) in Kombination mit einem Raspberry Pi und deinem Python-Tracking-Server in Betrieb zu nehmen, ist ein bestimmter Workflow nötig.

Wichtig vorab zur Architektur: Der Raspberry Pi übernimmt hierbei idealerweise den Part des **Abrufens und Entschlüsselns** der Standortdaten (mittels Python-Bibliotheken wie `FindMy.py`) und leitet diese an deinen Webserver weiter, da Openhaystack selbst für die initiale Schlüsselgenerierung und das Flashen des Tags zwingend einen **Mac** voraussetzt.

Schritt 1: Generierung der Schlüssel und Firmware (macOS erforderlich)

Das originale Openhaystack-Framework benötigt macOS, um die kryptografischen Schlüsselpaare (P-224 Elliptic Curve) zu erzeugen.

1. **Openhaystack App installieren:** Lade die offizielle Openhaystack-Anwendung von GitHub herunter.

2. **Accessory anlegen:** Erstelle in der App ein neues Accessoire (z. B. „Mein Pi-Tracker“). Die App generiert dabei ein privates/öffentliches Schlüsselpaar.

3. **Private Key sichern:** Der **Private Key** (wird im Mac-Keyring gespeichert) ist essenziell. Du *musst* diesen exportieren, da ihn später dein Python-Skript (auf dem Raspberry Pi oder dem Server) zum Entschlüsseln der Standort-Blobs benötigt.

4. **Firmware flashen:** Flashe die generierte Firmware samt integriertem Public Key auf dein Bluetooth-Zielgerät (z. B. ESP32 oder micro:bit). Das Gerät beginnt daraufhin, BLE-Signale (Find-My-Beacons) auszusenden, die von iPhones in der Nähe anonym aufgeschnappt und an Apple gemeldet werden.

Schritt 2: Vorbereitung des Raspberry Pi (Datenabruf)

Da macOS für den dauerhaften automatischen Abruf unpraktisch ist, übernimmt ein Raspberry Pi (oder ein beliebiger Linux-Server) das Polling der Daten über das Python-Ökosystem.

1. **Python-Umgebung einrichten:** Installiere Python (ab Version 3.8) und richte eine virtuelle Umgebung auf dem Raspberry Pi ein:

   ```
   python3 -m venv venv
   ```

```
source venv/bin/activate
```

2. **Bibliothek installieren:** Installiere die moderne Python-Bibliothek für das Find-My-Netzwerk (`FindMy.py`), welche plattformunabhängig ohne macOS-Abhängigkeiten funktioniert:

```
pip install findmy
```

Schritt 3: Implementierung des Abruf-Skripts auf dem Raspberry Pi

Erstelle ein Python-Skript (z. B. `tracker_sync.py`), das sich in regelmäßigen Intervallen bei Apple einloggt, die verschlüsselten Standortberichte für deinen Public Key abruft und mithilfe deines **Private Keys** entschlüsselt.

Beispielhaft sieht die Logik mit der `FindMy`-Bibliothek so aus:

```
import asyncio
from findmy import FindMyClient
from findmy.keys import AppleKeyPair

async def fetch_locations():
    # Credentials eines dedizierten (!!) Apple-Accounts
    async with FindMyClient(
        apple_id="dein-burner-account@icloud.com",
        password="dein-app-spezifisches-passwort"
    ) as client:

        # Deinen Private Key (aus Schritt 1) hier hinterlegen
        # priv_key = AppleKeyPair.from_hex("DEIN_PRIVATE_KEY_HEX")

        # Standorte abrufen
        reports = await client.get_reports()
        for report in reports:
            # Entschlüsselung mit dem Private Key
            # location = report.decrypt(priv_key)
            # print(f"Lat: {location.latitude}, Lon: {location.longitude}")
            pass

if __name__ == "__main__":
    asyncio.run(fetch_locations())
```

Schritt 4: Übertragung an den Tracking-Server (Web API)

Sobald das Skript auf dem Raspberry Pi neue Koordinaten entschlüsselt hat, sollen diese an deinen unternehmenseigenen Python-Tracking-Server (z. B. FastAPI, Flask oder Django) übermittelt werden.

1. **Schnittstelle am Tracking-Server einrichten:** Erstelle einen geschützten API-Endpoint (z. B. via HTTP POST mit einem Bearer-Token oder API-Key):

```
from fastapi import FastAPI, Header, HTTPException
from pydantic import BaseModel

app = FastAPI()

class LocationUpdate(BaseModel):
    device_id: str
    latitude: float
    longitude: float
    timestamp: int

@app.post("/api/v1/update_location")
async def receive_location(data: LocationUpdate, authorization: str = Header(None)):
    if authorization != "Bearer DEIN_GEHEIMER_API_TOKEN":
        raise HTTPException(status_code=401, Unauthorized="Invalid token")

    # Hier die Koordinaten in der Datenbank speichern
    print(f"Empfangen: {data.device_id} -> {data.latitude}, {data.longitude}")
    return {"status": "success"}
```

2. **Integration in den Raspberry-Pi-Job:** Erweitere das Skript aus Schritt 3 so, dass es nach erfolgreicher Entschlüsselung einen `requests.post()`-Befehl an deinen gehosteten Tracking-Server absetzt.

3. **Automatisierung (Cronjob / Systemd):** Richte auf dem Raspberry Pi einen `cronjob` ein, der dieses Skript beispielsweise alle 20 bis 30 Minuten ausführt:

   ```
   */30 * * * * /home/pi/venv/bin/python /home/pi/tracker_sync.py
   ```

Zusammenfassung der Voraussetzungen

- **Einmalig für Setup:** Ein Mac (macOS) mit installierter Openhaystack-App zum Generieren der Keys und Flashen der Hardware.
- **Die Hardware:** Ein physischer Openhaystack-Tag (z. B. ESP32).
- **Die Brücke:** Ein Raspberry Pi (oder ein kleiner Server daheim mit Heim-IP, um Apple-IP-Sperren von Cloud-Hostern zu umgehen) für das Abruf-Skript.
- **Apple-Konto:** Ein separater „Burner"-Apple-Account mit App-spezifischem Passwort für den automatisierten Abruf der Apple-Find-My-Server.
- **Schnittstelle:** Eine einfache REST-API (HTTP POST) auf deinem Tracking-Server, die die vom Raspberry Pi gelieferten JSON-Daten entgegennimmt und persistiert.

Sources

1. OpenHaystack: Build Your Own AirTags Today - Prompts
2. GitHub - seemoo-lab/openhaystack: Build your own 'AirTags' today
3. openhaystack/README.md at main · seemoo-lab ... - GitHub
4. FindMy.py: Query Apple's Network Without a Mac - Smart Converter
5. FindMy - PyPI
6. OpenHaystack - SourceForge

User prompt: Was genau ist das app-spezifische Passwort beim Einloggen als FindMy Client?

Response: Ein **app-spezifisches Passwort (App-Specific Password)** ist ein einmaliges, von Apple generiertes Passwort, das du für eine bestimmte Drittanbieter-Anwendung oder ein Skript verwendest, anstatt dein echtes Apple-ID-Passwort einzugeben.

Warum wird es benötigt?

1. **Sicherheit deines Hauptkontos:** Bibliotheken wie `FindMy.py` interagieren programmatisch mit Apples Servern. Wenn du dort dein normales Apple-ID-Passwort hinterlegen müsstest, wäre dieses im Klartext (oder in einer Konfigurationsdatei) auf deinem Raspberry Pi gespeichert. Sollte der Pi kompromittiert werden, hätte der Angreifer Zugriff auf deinen *gesamten* Apple-Account. Ein app-spezifisches Passwort begrenzt das Risiko.
2. **Umgehung der Zwei-Faktor-Authentifizierung (2FA):** Automatisierte Skripte können keine interaktiven 2FA-Abfragen (wie den 6-stelligen Code auf deinem iPhone) beantworten. Ein app-spezifisches Passwort signalisiert Apple: *"Diese Anwendung ist von mir autorisiert, sich anzumelden, ohne jedes Mal einen 2FA-Code abzufragen."*
3. **Einfaches Widerrufen:** Wenn du das Skript nicht mehr nutzt oder der Verdacht besteht, dass das Passwort abgefangen wurde, kannst du das app-spezifische Passwort einzeln bei Apple widerrufen, ohne dein Hauptpasswort ändern zu müssen.

Wie generiert man ein app-spezifisches Passwort?

Apple erlaubt die Erstellung solcher Passwörter nur für Konten, bei denen die Zwei-Faktor-Authentifizierung aktiviert ist.

1. Gehe auf die offizielle Webseite appleid.apple.com und melde dich mit deiner (Burner-)Apple-ID an.
2. Navigiere im Bereich **Anmeldung und Sicherheit** (Sign-In and Security) zum Punkt **App-spezifische Passwörter** (App-Specific Passwords).
3. Klicke auf das + (Plus-Zeichen), um ein neues Passwort zu generieren.
4. Gib einen Bezeichner ein (z. B. `FindMyPiTracker`), damit du später weißt, wofür dieses Passwort gedacht war.
5. Apple generiert einen Code im Format `xxxx-xxxx-xxxx-xxxx`.
6. Kopiere diesen Code und verwende ihn in deinem Python-Skript anstelle deines normalen Account-Passworts.
