Klar, hier ist ein ausführlicher Lernzettel, damit du offline Schritt für Schritt verstehst, wie du aus einem Passwort mit PBKDF2 einen Fernet-Schlüssel generierst und warum das wichtig ist.

Lernzettel: Aus Passwort einen sicheren Fernet-Schlüssel ableiten (offline, Schritt für Schritt)

---

1. Hintergrundwissen

- **Fernet** ist ein symmetrisches Verschlüsselungssystem aus der `cryptography`-Bibliothek.
- Für Fernet brauchst du einen **32 Byte langen Schlüssel**, der in **Base64** kodiert ist.
- Du kannst nicht einfach ein Passwort direkt als Schlüssel verwenden, weil:
    - Passwörter meist nicht 32 Byte lang sind.
    - Passwörter oft nicht zufällig genug sind.
- Deshalb wird ein **Key Derivation Function (KDF)** genutzt, der aus einem Passwort einen sicheren Schlüssel erzeugt.

---

2. Was ist PBKDF2?

- PBKDF2 (Password-Based Key Derivation Function 2) ist ein KDF, der:
    - Ein Passwort nimmt
    - Mit einem **Salt** (zufällige Bytes) kombiniert
    - Viele Iterationen rechnet (z.B. 100.000), um es rechnerisch teuer zu machen
    - Einen Schlüssel einer gewünschten Länge ausgibt (hier: 32 Byte)
- Das macht es Angreifern schwerer, das Passwort durch Brute Force zu knacken.

---

3. Warum Salt?

- Salt ist ein zufälliger Byte-String, der den Passwort-Hash einzigartig macht.
- Wenn du für jeden Nutzer/Anwendung ein anderes Salt nutzt, verhindert das, dass gleiche Passwörter zum gleichen Schlüssel führen.
- **Salt muss gespeichert werden!** Ohne den Salt kannst du später den Schlüssel nicht rekonstruieren.

---

4. Schritt-für-Schritt-Anleitung (offline)

Schritt 4.1: Passwort vorbereiten

- Ein sicheres Passwort wählen oder generieren, z.B. mit `secrets.token_urlsafe(16)`
- Beispiel:

```
password = "mein_super_sicheres_passwort"
password_bytes = password.encode()  # Für pbkdf2_hmac brauchen wir Bytes
```

Schritt 4.2: Salt erzeugen und speichern

- Salt ist eine zufällige Bytefolge, z.B. 16 Bytes lang.
- Beispiel:

```
import os
salt = os.urandom(16)
```

- Speichere `salt` in einer Datei oder Datenbank, damit du ihn wiederverwenden kannst.

Schritt 4.3: PBKDF2-Key ableiten

- Mit `hashlib.pbkdf2_hmac` (in Python Standardbibliothek)
- Parameter:
    - Hashfunktion: `'sha256'` (gut bewährt)
    - Passwort in Bytes
    - Salt
    - Iterationen, z.B. 100_000
    - Schlüsselgröße (dklen): 32 Byte für Fernet
- Beispiel:

```
from hashlib import pbkdf2_hmac
iterations = 100_000
key = pbkdf2_hmac('sha256', password_bytes, salt, iterations, dklen=32)
```

Schritt 4.4: Base64-Kodierung

- Fernet erwartet einen Base64-kodierten Schlüssel.
- Beispiel:

```
import base64
fernet_key = base64.urlsafe_b64encode(key)
```

- `fernet_key` ist jetzt der Schlüssel, den du für `Fernet(fernet_key)` nutzen kannst.

Schritt 4.5: Verschlüsseln und Entschlüsseln

- Beispiel:

```
from cryptography.fernet import Fernet
f = Fernet(fernet_key)

text = b"Geheime Nachricht"
token = f.encrypt(text)

decrypted = f.decrypt(token)
print(decrypted)  # Gibt b"Geheime Nachricht" aus
```

---

5. Wichtige Hinweise

- **Salt speichern!** Ohne den Salt kannst du den Schlüssel später nicht wiederherstellen.
- **Iterationen anpassen:** je mehr Iterationen, desto sicherer (aber langsamer). 100.000 ist ein guter Mittelwert.
- Verwende keine einfachen Passwörter. Nutze `secrets` oder lasse den Nutzer ein sicheres Passwort eingeben.
- Fernet-Schlüssel niemals als Klartext speichern, sondern nur Base64-codiert, so wie gezeigt.
- Nutze für echte Projekte eine sichere Methode, Salt und Passwort getrennt und sicher zu speichern.

---

6. Zusammenfassung: Ablauf

1. Passwort (vom Nutzer oder generiert) → Bytes
2. Zufälligen Salt erzeugen und speichern
3. PBKDF2 mit Passwort + Salt + Iterationen → 32 Byte Schlüssel
4. Schlüssel Base64-kodieren → Fernet Schlüssel
5. Fernet Schlüssel zum Ver- und Entschlüsseln verwenden

Wenn du das so befolgst, kannst du auch offline sicher Schlüssel aus Passwörtern ableiten und deine Daten verschlüsseln.

Falls du willst, kann ich dir den kompletten Ablauf auch als kommentiertes Beispielskript schreiben!