Hochschule Worms
Fachbereich Informatik
Studiengang Angewandte Informatik - dual (M.Sc)

Projektbericht
Deep Dive

Entwicklung einer Lösung zur Berechtigungsverwaltung von Secrets zwischen Entwicklerinnen und Entwicklern

In der Arbeitsumgebung des Partnerunternehmens Medienagenten oHG

Version 1.1

Vorgelegt von

Leon Etienne, 676838
inf4437@hs-worms.de
Im Wintersemester 2024/25

bei Professor Dr. Heinemann
heinemann@hs-worms.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist noch nicht in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung vorgelegt worden.

Ludwigshafen am Rhein, 14. Februar 2025

Leon Etienne _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1PSA 1Password Secrets Automation

1P 1Password

API Application Programmer Interface

CLI Command Line Interface

GAU Größter Anzunehmender Unfall

GUI Graphical User Interface

MASA Medienagenten Secret Authority

UUID Universally Unique Identifier

YAML Yet Another Markup Language

Glossar

(1P-)Eintrag/Secret

Eine Gruppierung an Daten, die einen Login ermögliche. Z.B. (Nutzername, Passwort). Eine solche Struktur kann bein 1P aus beliebig vielen Schlüsselwertpaaren bestehen. Wird in dieser Ausarbeitung synonym zu 'Secret' verwendet.

(1P-)Vault

Eine Kollektion an Secret-Einträgen in einem Passwort-Manager (1Password).

Ansible-Playbook/s

Ansible-Playbooks sind Skripte, mit dem Ziel einen deklarierten Zustand herzustellen. [Red Hat, Inc., 2025a]

Ansible-Role

Eine Abfolge von definierten Schritten, die von Ansible-Playbooks ausgeführt werden. Eine solche Role könnte z.B. einen Server buchen, konfigurieren und verwendungsfertig bereitstellen.

Detailansicht / Detailaufruf

Eine Anfrage, *ein* Objekt oder *einen* Eintrag zurückgibt oder manipuliert.

Docker

Eine arrivierte Container-Engine für Anwendungsentwicklung.

Entwickler*innen-Vault

Ein 1P-Vault, in dem die Kopien bzw. Referenzen auf Einträge des Partnerunternehmens verortet sind. Hier liegen nur Kopien bzw. Referenzen! Der Sinn eines solchen Vaultes ist es, Entwickler*innen beschränkten Lesezugriff auf Daten der Nutzvaults zu gewähren.

Jinja-Templating-Engine

Eine Templating Engine, die das Jina-Format verwendet. Ansible

verwendet eine Jinja-Templating-Engine.

Listenansicht / Listenaufruf

Eine Anfrage, die eine Liste von Objekten oder Einträgen zurückgibt.

Nutzvault

Ein 1P-Vault, in dem die originalen Einträge des Partnerunternehmens verortet sind. Hier liegen keine Kopien, sondern Originale!

Öffentliche 1P-UUID

Die 1P-UUID, die zu einem originalen Eintrag gehört, und nicht zu einer Referenz. Externe Entwickler*innen haben also keinen direkten Zugriff auf einen solchen Eintrag, sondern müssen stattdessen eine Referenz auf diesen Eintrag verwenden. Eine solche UUID kann beispielsweise in Ansible-Konfigurationdateien stehen und von jedem*r Entwickler*in verwendet werden.

Private 1P-UUID

Die 1P-UUID, die zu einem Referenz-Eintrag gehört, und nicht zu einem originalen Eintrag. Diese Einträge sind emphemerisch und existieren ausschließlich in den einzelnen Developer-Vaults und sind Entwickler*innen-spezifisch.

Secret

Eine vertrauliche Information, wie zum Beispiel ein Passwort.

Toolbox

Eine Ansammlung an Werkzeugen, wie zum Beispiel Skripte.

1. Einleitung

1.1. Problemstellung

In der Arbeitsumgebung des Partnerunternehmens besteht zum Zeitpunkt der Themenfindung der hier beleuchteten Arbeit kein Management für Secrets und Logindaten zwischen Entwickler*innen. Logindaten zu den Projekten des Unternehmens liegen schlicht in einem 1P-Vault. 1P ist der vom Unternehmen verwendete Passwortmanager. Auf diesen Vault haben sämtliche internen Entwickler*innen Zugriff, jedoch keine externen Entwickler*innen. Der Grund dafür ist, dass anderenfalls Lesezugriff auf sämtliche Einträge dieses Vaults gegeben werden müssten. 1P unterstützt keine Freigaben einzelner Einträge an andere Nutzer, ohne diese Einträge in einen eigenen Vault zu kopieren. Würden diese manuell in einen eigenen Vault kopiert werden, müssten diese Einträge fortan redundant gepflegt werden. Das ist eine Fehlerquelle, die zu asynchronen Einträgen führt. Außerdem ist das ein großer Arbeitsaufwand. All das gestaltet das Einbinden von externen Entwickler*innen, wie z.B. Freelancer*innen, schwer.

Ein weiteres Problem ist, dass Secrets in Konfigurationsdateien, die firmeninternen Ansible-Scripten beilegen, unverschlüsselt einsichtig sind. Das macht es zu einem großen Sicherheitsrisiko und somit inpraktikabel externen Entwickler*innen Zugriff auf dieses Ansible-Repository zu gewähren. Dieses Ansible-Repository ist jedoch zwingend erforderlich, um eine Entwicklungsungebung für Firmenprojekte auf dem lokalen Rechner zu schaffen. Auch hier sind Lösungen für externe Entwickler*innen zumeist unschöne Workarounds.

1.2. Zielsetzung

Ziel ist es, eine Umgebung zu schaffen, in der beliebigen Entwickler*innen bestimmte 1P-Einträge zugewiesen werden können. Der Pflegeaufwand sollte hierbei überschaubar bleiben. Das heisst, dass z.B. ganze Gruppen von Einträgen Entwickler*innen zugewiesen werden können. Wenn z.B. einem Projekt viele Einträge zugeordnet sind, sollten diese idealerweise mit einer einzigen Konfigurationszeile einem*r Entwickler*in zugeordnet werden können. Außerdem sollte eine Möglichkeit ausgearbeitet werden, um 1P-Einträge in Ansible auszulesen, damit keine Secrets mehr in den beiliegenden Konfigurationsdateien stehen, die das Freigeben dieser zu einem Sicherheitsproblem machen.

1.3. Methodische Vorgehensweise

Einige Anforderungen sind bereits im Voraus definiert. Weiterführende Anforderungen werden im Rahmen einer Anforderungserfassung ermittelt. Anschließend werden verschiedene Lösungsansätze betrachtet und auf Tauglichkeit geprüft. Nachdem ein akzeptabler Lösungsweg gefunden ist, wird dieser umgesetzt. Abschließend wird der Erfolg des Unterfanges evaluiert und mögliche, auf dieses Projekt aufbauende Arbeiten in Ausblick gestellt.

2. Grundlagen

2.1. Die Arbeitsumgebung

Die Arbeitsumgebung des Partnerunternehmens besteht für diese Themenstellug nennenswert aus:

- Cloudbasierten Web- und Datenbankservern
- Git-Repositories bei Bitbucket
- Der lokalen, Docker-basierten Arbeitsumgebung
- Ein Ansible-Playbook, das ein Projekt mit Daten aus der Cloudumgebung und Code aus Bitbucket in der lokalen Entwicklungsumgebung bereitstellt.

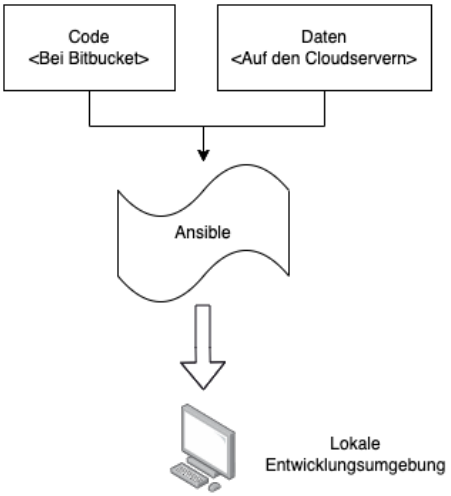


Abbildung 1.: Relationsdiagramm: Bereitstellen von Projekten des Partnerunternehmens in einer Entwicklungsumgebung

Quelle: Eigene Darstellung

Die lokalen Arbeitsumgebungen der Entwickler*innen liegen großteils außerhalb des Firmennetzwerkes, da diese Entwickler*innen oft oder

ausschließlich im mobilen- bzw, Homeoffice arbeiten. Ein Firmen-VPN-Netz existiert nicht und ist auch nicht erwünscht.

2.2. 1Password

1P ist der vom Partnerunternehmen verwendete Passwort-Manager. Bereits vor Beginn der Bearbeitung dieser Themenstellung wurde deutlich gemacht, dass es Ziel ist, 1P auch für das Verwalten von Secrets in Ansible zu verwenden.

2.3. Ansible

Ansible ist ein Automatisierungswerkzeug von Red Hat, Inc. und hat das Ziel, einen definierten Zustand im behandelten System herzustellen. [Red Hat, Inc., 2025a] Ein Administrator definiert also nicht die erforderlichen Schritte, um einen Zustand z zu erreichen, sondern lediglich z selbst. Ansible kann über speziell gefertigte Python-Module um Schnittstellen erweitert werden.

3. Anforderungen

3.1. Anforderungserfassung

Obwohl bereits vor Beginn des Projektes einige Anforderungen bekannt sind, müssen manche Details nachträglich in Erfahrung gebracht werden. Hierfür wurde ein informales Interview mit dem Stakeholder durchgeführt. Im Rahmen dieses Interviews wurden vorbereitete Fragen gestellt, dem Stakeholder aber auch die Möglichkeit gegeben frei heraus zu sprechen und Wünsche zu äußern. Notizen zu diesem Interview befinden sich im Anhang unter *⟨⟨A Stakeholder-Interview⟩⟩*.

3.2. Ergebnisse

Das Ergenis der Anforderungserfassung ist ein Lastenheft, das in constraints, funktionale und nicht-funktioniale Anforderungen unterteilt ist. Im Zuge des Interviews und diversen anderen, informalen Gespräche, hat sich der Autor ein tiefes Verständnis für das vorliegende Problem des Auftraggebers anggeignet. Das untenstehende Lastenheft wurde mit dem Stakeholder besprochen und bestätigt.

Funktionale Anforderungen
Entwickler*innen erhalten verschiedene Zugang zu verschiedenen 1P-Einträgen (Zugänge), definiert in einer YAML-Datei.
Wildcard-Matching auf den 1P-Eintragstitel für zusammenhängende Einträge.
1P-Einträge sollen Entwickler*innen einzeln zuweisbar sein.
Nicht in der Konfugration gelistete Zugänge sollen bei Anwendung entfernt werden.
Ansible-Secrets müssen aus 1P dereferenziert werden können.
Einträge sollen für Entwickler*innen einsehbar sein.
Nicht-funktionale Anforderungen
Das System muss Berechtigungen von Entwickler*innen verwalten.
Das System muss benutzerfreundlich sein.
Das System darf nicht aufwändig zu pflegen sein.
Die benötigte Zeit zur Ausführung der Anwendung soll nicht sehr lange sein.
Das System muss robust gegenüber Misskonfigurationen sein, die zur Löschung der zugrunde liegenden 1P-Einträgen führen könnten.
Constraints
Nutzung von 1P ist zwingend erforderlich.
Die Übermittlung der Secrets muss über das Internet erfolgen.

Tabelle 1.: Lastenheft

4. Technische Umsetzung

4.1. Berechtigungsverwaltung

4.1.1. Ausarbeitung der Herangehensweise

Zunächst wurde gebrainstormed, welche Herangehensweisen hier möglich sind. Ein Artefakt des Brainstormings ist eine Mind-Map, die unter *⟨⟨B Ideensammlung⟩⟩* zu finden ist.

Ansatz 1

Der aus dieser Mindmap, nach individueller Meinung des Autors, vielversprechenste Ansatz ist es, die 1P-Restful-API zu verwenden. Bei diesem Ansatz würden Administrator*innen und Entwickler*innen API-Keys für 1P erhalten. Entwickler*innen hätten mit ihren Keys bestimmte Leseberechtigungen r und Administratoren die Berechtigung r zu verändern.

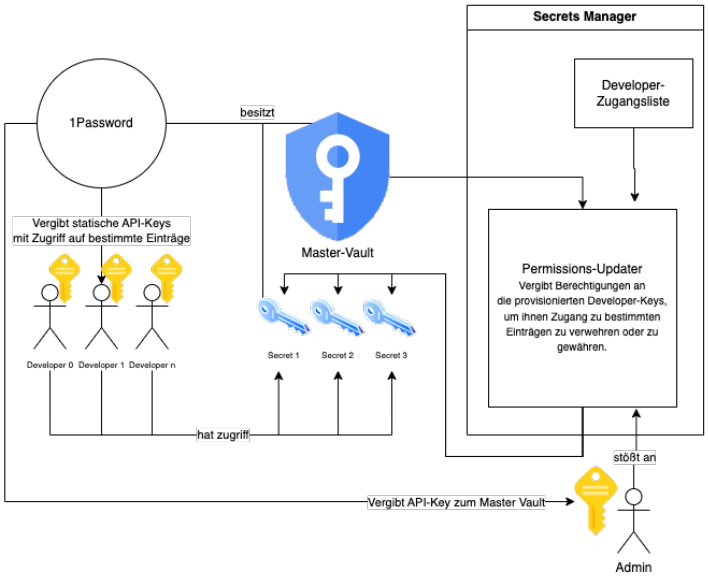


Abbildung 2.: Relationsdiagramm: Ansatz 1 | 1Password-API

Quelle: Eigene Darstellung

Dieser Ansatz wurde zeitnah als unumsetzbar erkannt und verworfen, da 1P das nachträgliche Verändern von API-Key-Berechtigungen nicht erlaubt.

Ansatz 2

Der nächste Lösungsansatz befasst sich mit einer Abstraktionsebene: Der Medienagenten Secret Authority (MASA). Hier ist die grundlegende Idee, dass es eine serverseitige Anwendung gibt, die sich MASA nennt. Diese Anwendung übernimmt die Aufgabe anhand eines hinterlegten 1P-API-Keys Secrets aus dem 1P-Vault des Partnerunternehmens abzufragen und an Entwickler*innen weiterzureichen. Die MASA provisioniert eigene API-Keys an Entwickler*innen und vermermerkt serverseitig, welcher API-Key berechtigt ist, welche 1P-Einträge abzufragen. Der API-Key könnte grundlegende Informationen wie zum Beispiel Entwickler*innennamen und Ablaufzeitpunkte des Keys einbetten. Dieser Ansatz trägt viel Sicherheitsverantwortung, da eine mögliche Ausnutzung

einer Sicherheitslücke der MASA direkt in den Firmen-Passwortmanager führen würden. Um diesem Risikofaktor entgegenzuwirken würde der 1P-Key der MASA verschlüsselt werden und die MASA würde nur einen Teil des Entschlüsselungs-Keys vorrätig halten. Der andere Teil wäre in jedem Entwickler*innen-Key eingebettet. Dadurch wäre gewährleistet, dass ein*e Angreifer*in, selbst bei sehr weitreichendem Zugriff in die MASA, nicht auf das Innere des Passwortmanagers zugreifen könne, da die MASA dazu selbstständig gar nicht im Stande wäre. Da Entwickler*innen lediglich ein Schlüsselfragment des Verschlüsselungs-Schlüssels in ihrem Key eingebettet hätten, der einen serverseitigen Schlüssel der MASA zum auslesen benötigt, bestünde auch keine Gefahr, dass ein*e Entwickler*in anhand seines bzw. ihres Keys ungeschützten Zugang zum Passwortmanager erhalten würde. Dieser Ansatz erlaubt für weitreichende Flexibilität, da sämtliche Logik, die sich mit Berechtigungen beschäftigt, anwendungsfallspezifisch geplant und umgesetzt wäre.

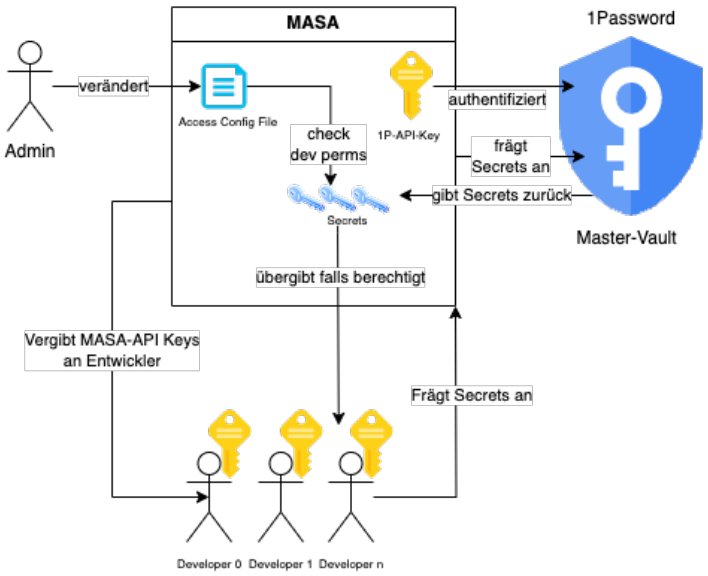


Abbildung 3.: Relationsdiagramm: Ansatz 2 | MASA

Quelle: Eigene Darstellung

Letztendlich entschied sich der Stakeholder gegen die Umsetzung der

MASA, da dieser Ansatz für zu Aufwändig betrachtet wird und für den durch sie erbrachten Vorteil zu viel Aufwand und Angriffsfläche schaffen würde.

Ansatz 3

Der letzte Lösungsansatz befasst sich mit dem Erstellen dedizierter Vaults für jede*n Entwickler*in *d*. Hierbei existiert eine Python-Toolbox, die anhand einer Yaml-Datei Referenzen auf diese Passwort-Einträge in Vault_d legt und von dort entfernt, wenn ein solcher Zugriff laut der Yaml-Datei nicht mehr vorgesehen ist. Diese Einträge können über feste Eintrags-IDs und über Regex bezogen auf die Eingrags-Titel einem/r Entwickler*in vorgesehen werden.

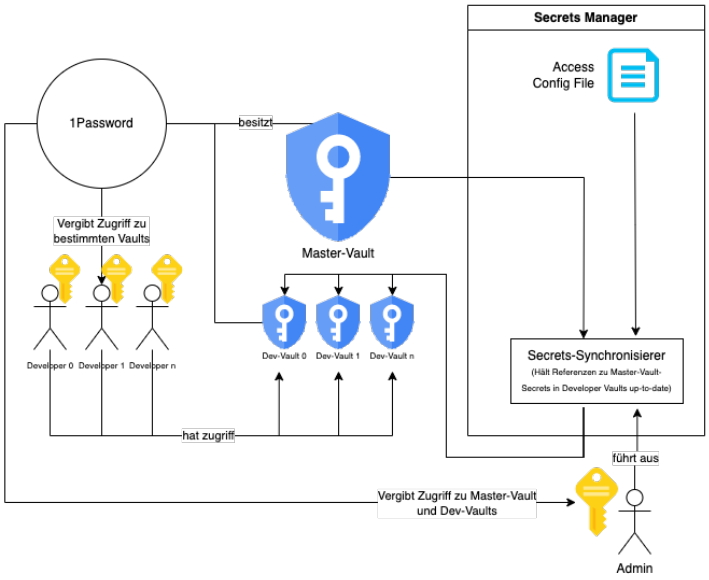


Abbildung 4.: Relationsdiagramm: Ansatz 3 | Python-Toolbox

Quelle: Eigene Darstellung

Letztendlich entschied sich der Stakeholder für Ansatz 3, da er ihm kostengünstig und aursreichend erschien.

4.1.2. Kodierung

Um den vom Stakeholder ausgewählten Ansatz 3 wie geplant umzusetzen, wurden zunächst die Dokumentationen diverser 1P-Schnittstellen konsultiert. Schnell offenbarte sich eine Alternative zu API-Keys: Die 1P-Desktop-Anwendung stellt eine CLI-API bereit. Die CLI-API der Desktop-Anwendung zu verwenden, würde drei Probleme lösen:

Kosten und hedonische Qualität

Einen API-Key zu erstellen und zu übermitteln ist kostenspielig und umständlich. Ein 1P-Konto haben dem gegenüber bereits alle Entwickler*innen des Partnerunternehmens.

Authentifizierung und Sicherheit

Anstatt einen API-Key unsicher zu speichern und in relevante Programme (=Ansible) zu laden, wird die Authentifizierung zu 1P ausgelagert.

Manuelle Einsicht

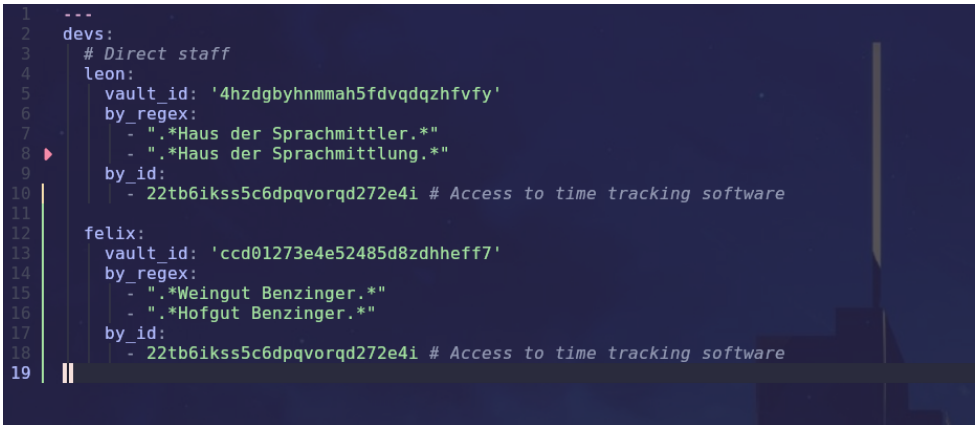
Da über die CLI-Methode der Zugriff auf die Entwickler*innen-Vaults direkt über die 1P-Desktop-App geschieht, kann diese den Vault-Inhalt auch dem Nutzer in ihrem GUI offenbaren.

Integrationsaufwand

Die Verwendung der 1P-Restful-API erscheint dem Autor nach ihrer Dokumentation sehr aufwändig und kompliziert. Eine CLI-API zu verwenden würde somit in der Umsetzung Projektressourcen sparen.

So begründet fällt die Wahl der Schnittstelle zu 1P auf ihre CLI-API. Um schnelle Softwareentwicklung mit minimalem Overhead zu gewährleisten und um für eine spätere Einbindung in Ansible bereits in Vorleistung zu treten, fällt die Wahl der Programmiersprache auf Python. Ansible-Module können mit Python geschrieben werden. [Red Hat, Inc., 2019] Es wurde eine rudimentäre Architektur entworfen, die beschreibt, welche Komponente des Werkzeuges aus welchen kleineren Komponenten

besteht. Am unteren Ende dieses Aggregatbaumes stehen atomare Operationen. Im Kontext dieses Werkzeuges sind atomare Operationen Operationen, die vom 1P-CLI ausgeführt werden. Diese Operationen implementiert also 1P selbst. Hierbei handelt es sich nur um Lese, Erstell- und Löschvorgänge. Das Erfassen, auf welchen Eintrag welche*r Entwickler*in Zugriff hat, und auf welche nicht, übernimmt *sync-dev-vault.py*. Die Funktionen der andere Skripte ergeben sich in Gänze aus ihren Dateinamen.



```
---
devs:
  # Direct staff
  leon:
    vault_id: '4hzdgbyhnmmah5fdvqdqzhfvfy'
    by_regex:
      - ".*Haus der Sprachmittler.*"
      - ".*Haus der Sprachmittlung.*"
    by_id:
      - 22tb6ikss5c6dpqvorqd272e4i # Access to time tracking software

  felix:
    vault_id: 'ccd01273e4e52485d8zdhheff7'
    by_regex:
      - ".*Weingut Benzinger.*"
      - ".*Hofgut Benzinger.*"
    by_id:
      - 22tb6ikss5c6dpqvorqd272e4i # Access to time tracking software
```

Abbildung 5.: Struktur der Zugriffs-config.yml

Quelle: Eigene Darstellung

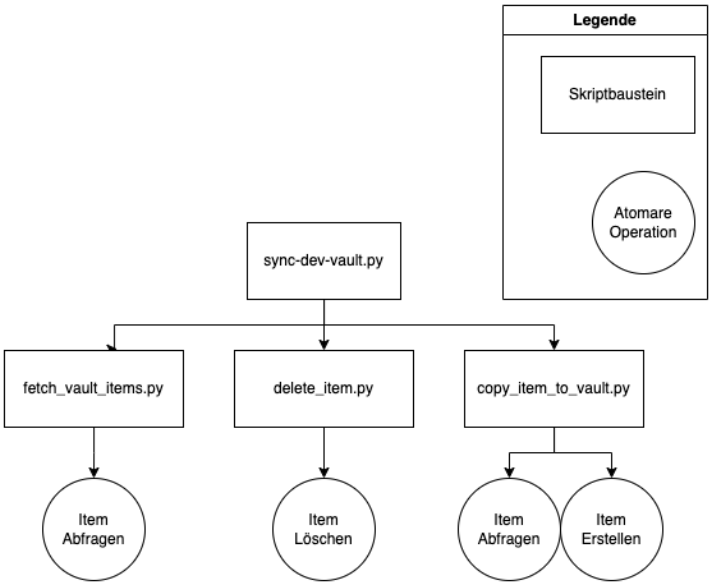


Abbildung 6.: Diagramm: Programmstruktur Secret-Synchronizer

Quelle: Eigene Darstellung

Die Funktionsweise des Programmes ist wie folgt:

1. Wende die Schritte 2..n auf alle zu synchronisierenden Entwickler*innen $d \in D$ an.
2. Lösche alle Einträge $e_d \in d$.
3. Kopiere alle vorgesehenen Einträge $e_m \in$ Master-Vault nach d und ergänze das Feld “Originale Eintrags-ID” mit dem Universally Unique Identifier (UUID) von e_m.
4. Erstelle ein ID-Mapping-Objekt von e_d zu e_m in d.

Performanzprobleme und Optimierung

Eine Schwierigkeit, die sich im Rahmen der Umsetzung offenbart hat, ist, dass das 1P-CLI sehr langsam ist. Soll die Liste aller Einträge eines Vaults ausgelesen werden, dauert das nach Erfahrungen des Autors

etwa eine Sekunde, da es sich hierbei um *eine* Anfrage handelt. Diese Listenansicht zeigt jedoch nur begrenzte Informationen der Einträge. Soll ein bestimmtes Feld eines Eintrages (z.B. “Originale Eintrags-ID”) in Erfahrug gebracht werden, müssen alle Informationen *eines* Eintrages abgefragt werden. Hier ist *ein* CLI-API-Aufruf pro Eintrag erforderlich. Sind einem*r Entwickler*in z.B. 30 Einträge zugeordnet, so dauert das Finden eines Eintrages mit einer bestimmten originalen ID im Durchschnitt $n = 30; \frac{n}{2} = 15$ Sekunden. Im langsamsten Fall wären es $n = 30$ Sekunden. Ein Kopiervorgang stellt zwei Aufrufe dar (=*lesen,erstellen*), also dauert das Kopieren von 30 Einträgen $n = 30; 2n = 60$ Sekunden. Das 1P-CLI kann zwar für Detail-Aufrufe mehrere Eintrags-IDs auf Standard-Input annehmen und bearbeiten, jedoch zeigen Versuche des Autors, dass das nicht die benötigte Zeit pro Eintrag beeinflusst. Das heißt, eine Anfrage für zehn Einträge zu stellen, dauert in etwa zehn mal so lange, wie zehn Anfragen für jeweils einen Eintrag zu stellen.

Die naive Herangehensweise, um einen Eintrag einer bestimmten originalen UUID zu finden, ist es, jeden Eintrag auf dessen originale UUID zu prüfen, bis der passende Eintrag gefunden wurde. Diese Lösungsweg hat eine Zeitkomplexität von $O(n^2)$. Eine spätere Ergänzung, um die programmatische Auslesung der Einträge einer bestimmten originalen UUID in $O(n)$ anstatt $O(n^2)$ zu gewährleisten, ist die Unterhaltung von Mapping-Objekten in Entwickler*innen-Vaults. Je Entwickler*innen-Vault wird abschließend der Synchronisierung ein Mapping-Objekt erstellt. Diese Mapping-Objekte halten die Informationen vorrätig, welche öffentliche 1P-UUID zu welcher privaten 1P-UUID gehört. Mit diesen Mapping-Objekten kann ein beliebiger Eintrag anhand einer öffentlichen UUID binnen zwei Anfragen erfasst werden: Anfragen des Mapping-Objektes und Anfragen des privaten Objektes. Das entspricht einer Zeitkomplexität von $O(2n) = O(n)$. Desweiteren kann 1P Eintragsdaten lokal zwischenspeichern. Diese Option lässt sich mit dem Flag *–cache* auf Leseoperationen anwenden und beschleunigt das Auslesen der angefragten Informationen, soweit diese im Cache existieren. Auf Unix-

ähnlichen Betriebssystemen ist dieses Verhalten standardmäßig aktiviert. [1Password, 2025b]

Sicherheitsbedenken

Die Konfigurationsdatei definiert Entwickler*innen-Vaults, über ihre UUIDs. In diesen IDs sieht ein*e Administrator*in keine Vaultnamen. Diese sind nicht sprechend. Wenn nun aus etwaigen Gründen dort die UUID eines Nutzvaults des Partnerunternehmems aufgeführt wäre, würde das Werkzeug alle sich dort befindlichen Zugänge löschen. Das wäre ein Super-GAU in Form von Datenverlust.

Sicherheitsvorkehrungen

Um das zu verhindern, wurde eine Liste mit wichtigen Vault-IDs fest einkodiert. Alle Erstell- oder Löschmethoden müssen einen Vault-ID-Parameter erhalten, selbst wenn dieser technisch nicht notwendig ist. Wenn diese Vault-ID nun in der Liste der fest kodierten Nutzvault-IDs vorkommt, meldet die Methode einen deskriptiven Fehler und beendet die Programmausführung. Somit ist gewährleistet, dass selbst bei einer fatalen Fehlkonfiguration kein Datenverlust entseht.

4.2. Integration in Ansible

Eine Anfordergung beschreibt, dass 1P-Einträge von Entwickler*innen innerhalb von Ansible-Playbooks dereferenziert und verwendet werden können.
1P unterstützt nativ das Ersetzen von 1P-Referenzen in Dateien durch Secrets. Diese Technik nennt sich “1Password Secrets Automation (1PSA)”. Diese Technologie ist jedoch nicht für die hier vorliegende Aufgabenstellung verwertbar, da die dem zugrunde liegende Berechtigungsverwaltung auf Vault-Basis steht. Entweder hat ein*e Entwickler*in Zugriff auf einen gesamten Vault, oder er*sie keinen Zugriff auf den gesamten Vault. Eine feingranularere Steuerung ist hier nicht möglich, jedoch für die hier gegebenen Anforderungen nötig. [1Password, 2025a] 1PSA erfasst UUIDs

und ist daher mit dem Konzept von Entwickler*innen-Vaults inkompatibel, da Entwickler*innen hierbei eigene Kopien der originalen Einträge führen, die jeweils eigene UUIDs haben. Externe Entwickler*innen haben somit keinen Zugriff auf die originalen, öffentlichen UUIDs und die privaten UUIDs, die im Entwickler*innen-Vault vorhanden sind, gelten jeweils nur für ein*e Entwickler*in. Das erfordert eine maßgeschneiderte, programmatische Lösung:

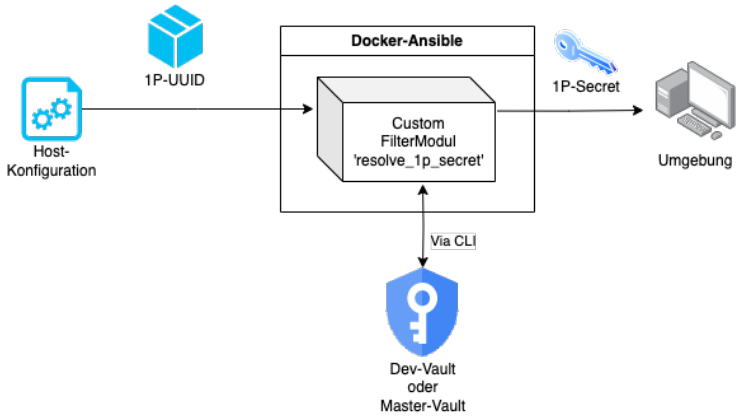


Abbildung 7.: Relationsdiagramm: Docker-Ansible-Struktur, um 1P-Einträge zu dereferenzieren

Quelle: Eigene Darstellung

Die hierfür angedachte Architektur erfordert, dass öffentliche UUIDs in Host-Konfigurationen in Docker-Ansible aufgeführt sind. Also eine UUID, die für alle Entwickler*innen greifbar ist.
Ab hier wird die Nutzergruppe “Entwickler*innen” in zwei Untergruppen strukturiert:

Interne Entwickler*innen

Interne, festangestellte Entwickler*innen haben Vollzugriff auf den 1P und somit auch Zugriff auf die in den Host-Konfigurationen vermerkte, öffentliche UUID eines Eintrages. Da diese Entwickler*innen keinen Entwickler*innen-Vault haben, müssen sie direkt auf diese notierte, öffentliche UUID zugreifen.

Externe Entwickler*innen

Externe Entwickler*innen verfügen über einen Entwickler*innen-Vault, nicht jedoch über direkten Zugriff auf die vermerkte, öffentliche UUID. Falls der*die jeweilige Entwickler*in Zugriff auf einen verlinkten Eintrag hat, dann nur auf eine Kopie des Eintrages in dessen*deren jeweiligen Entwickler*innen-Vaults. Diese Kopie hat eine andere UUID als die, die in der Host-Konfiguration steht. Sie ist ja auf technischer Ebene ein anderer Eintrag, nur mit identischem Inhalt. Die in den Host-Konfigurationen vermerkten, öffentlichen UUIDs müssen also zunächst in eine private, sich im Entwickler*innen-Vault befindliche, UUID übersetzt werden.

Um diese Problemstellung anzugehen, wird ein Ansible Filtermodul entworfen. Ein Filtermodul dient als Texttransformator und kann in Ansible verwendet werden.

`{{ "hello world" | uppercase }}`.
[Red Hat, Inc., 2025b] Dieses Beispiel führt das “uppercase”-Filtermodul an. Ein Beispiel mit dem im Rahmen dieser Ausarbeitung bereitgestellten Filtermodul würde so aussehen:
`{{ smtp.password | resolve_1p_secret }}`.

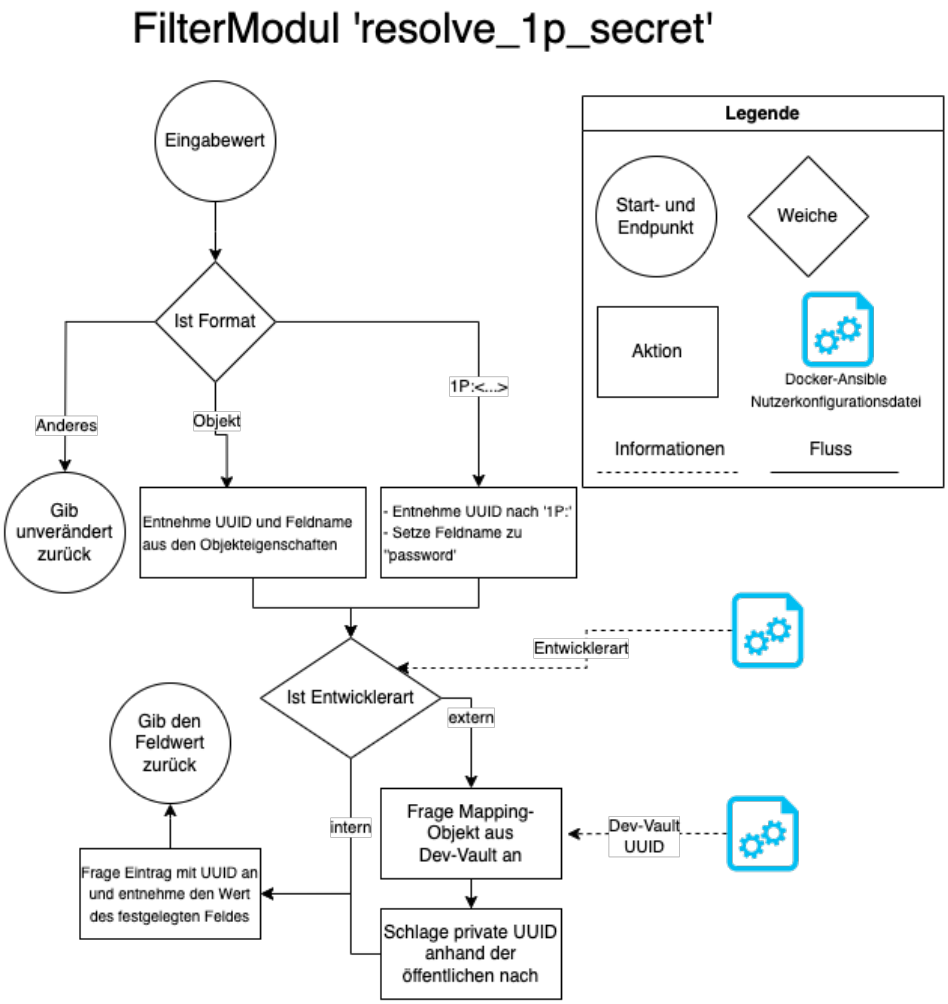


Abbildung 8.: Flussdiagramm: Businesslogik des 1P-Resolver-Filtermoduls

Quelle: Eigene Darstellung

4.2.1. Akzeptierte Formate

Das Filtermodul akzeptiert mehrere, verschiedene Eingabeformate und ist rückwärtskompatibil.

Kein erkanntes Format

Wird kein bekanntes Format erkannt, wird der Wert unverändert zurückgegeben. Das ermöglicht Rückwärtskompatibilät, sodass nach-wie-vor hardgecodede Secrets funktionieren, und nicht versehentlich als UUID interpretiert werden. Das gewährleistet eine flüssigere Migration der Host-Konfigurationen, da somit bestehende Dateien weiterhin valide sind.

1P:<...>

Beginnt der Wert mit "1P:", so wird alles Nachfolgende als UUID interpretiert. Es wird versucht, das Feld "password" aus diesem Eintrag zu dereferenzieren.

Objektformat

Wird ein Yaml-Objekt übergeben, so werden die Keys "1P_secret_uuid" und "1P_field_id" erwartet. Diese definieren die Werte der Eintrags-UUID und der Feld-ID in der ein Secret steht. Das ermöglicht z.B. auch den Benutzernamen (*1P_field_id: username*) eines Eintrages abzufragen, anstelle nur des Passworts. Ist keine Feld-ID gegeben, so wird auf das Standardfeld "password" zurückgefallen.

4.2.2. Übersetzung der UUIDs

Um die private UUID zu ermitteln, die zu der öffentlichen UUID gehört, die in der Host-Konfiguration steht, frägt das Filtermodul den Mapping-Eintrag aus dem jeweiligen Entwickler*innen-Vault an und schlägt die öffentliche UUID darin nach. Die daraus resultierende UUID kann von einem*r externen Entwickler*in angefragt werden.

4.2.3. Unterscheidung zwischen internen und externen Entwickler*innen

Ist in der Docker-Ansible-Konfigurationsdatei eine Entwickler*innen-Vault-UUID definiert, so wird von einem*r externe*n Entwickler*in ausgegangen. Ist stattdessen definiert, dass die in den Host-Konfigurationen angegebenen, öffentlichen UUIDs direkt angefragt werden sollen, wird von einem*r interne*n Entwickler*in ausgegangen. Sind beide Konfigurationen zugleich gegeben, wird ein Inkompatibilitätsfehler erhoben. Ist ein*e interne*r Entwickler*in angenommen, so wird der Mapping-Schritt übersprungen. Der verbleibende Prozess bleibt unberührt.

4.2.4. Kommunikation mit 1Password

Ist eine UUID ermittelt, auf die der*ie Nutzer*in Zugriff hat, wird diese über das 1P-CLI angefragt. Dieser Aufruf ist: *op item get <UUID>* mit dem Zusatz *–format json*, um die Ausgabe programmatisch auswerten zu können.

4.2.5. Performanz und Benchmarks

Um diese Konfiguration zu testen, werden in einem Testszenario fünf Werte aus 1P ausgelesen:

- Datenbank-Host
- Datenbank-Port
- Datenbank-Benutzername
- Datenbank-Passwort
- Datenbank-Name

Diese Einträge abzufragen dauert durch das imperformante 1P-CLI rund 17 Sekunden.

4.2.6. Optimierung

Es bieten sich zwei Möglichkeiten an, den in *⟨⟨8 Flussdiagramm: Businesslogik des 1P-Resolver-Filtermoduls⟩⟩* abgebildeten Prozess zu beschleunigen. Diese beschäftigen sich damit, zu limitieren, wie oft das 1P-CLI angefragt wird:

Das Mapping-Objekt zwischenspeichern

Anstatt das Mapping-Objekt für jedes angefragte Secret erneut abzufragen, könnte es zwischengespeichert werden.

Ganze Einträge zwischenspeichern

Für den Fall, dass ein Eintrag mehrach angefragt wird, könnte ein einmal angefragter Eintrag zwischengespeichert werden. Würden mehrere Felder eines Eintrages angefragt werden (z.B. *=Host, Port, Benutzername, Passwort, Name*), so müsse der gesamte Eintrag nur ein mal angefragt werden.

All diese Ansätze erfordern es Daten zwischenzuspeichern. Filtermodule haben allerdings keinen persistenten Arbeitsspeicher über verschiedene Aufrufe hinaus. E.g. nach jedem angefragten Secret, verliert er seinen internen Zustand. Daher muss dieses zwischengespeichert in einer Datei stattfinden. Hierfür wird eine Tempdatei über Pythons *tempfile.gettempdir()*-Funktion ermittelt. Diese Zwischenspeicher beinhalten die vom 1P-CLI zurückgegebenen Informationen.

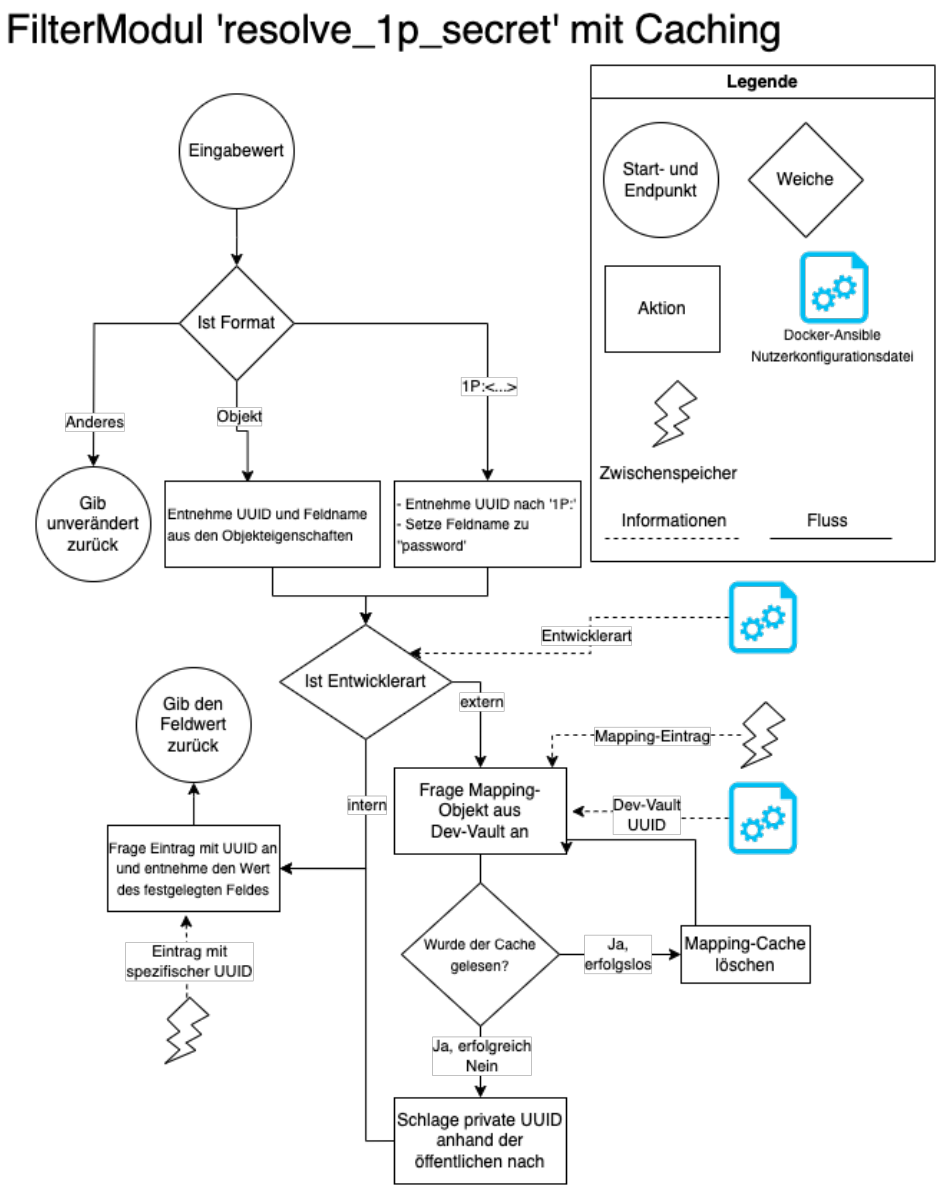


Abbildung 9.: Flussdiagramm: Businesslogik des 1P-Resolver-Filtermoduls, mit Zwischenspeicher

Quelle: Eigene Darstellung

Mapping-Zwischenspeicher

Wenn das Mapping-Objekt angefragt wird, so wird zunächst geprüft, ob es eine lokal zwischengespeicherte Version gibt. Falls ja, wird diese geladen und verwendet. Falls nein, wird es via dem 1P-CLI angefragt und lokal gespeichert. Sollte der Mapping-Zwischenspeicher verwendet werden und das gesuchte Objekt nicht darin gefunden werden, so wird dieser Zwischenspeicher gelöscht und das Mapping-Objekt erneut angefragt. Das deckt folgenden Problemfall ab:

1. Zwischenspeicher wird erstellt
2. Entwickler*in erhält mehr Berechtigungen
3. Entwickler*in will diese Berechtungen nutzen
4. Der Zwischenspeicher wird verwendet, spiegelt diese neuen Berechtigungen aber nicht wider
5. Der Prozess schlägt zu Unrecht fehl

Dieser Prozess würde durch diese Vorkehrung so aussehen:

1. Zwischenspeicher wird erstellt
2. Entwickler*in erhält mehr Berechtigungen
3. Entwickler*in will diese Berechtungen nutzen
4. Der Zwischenspeicher wird verwendet, spiegelt diese neuen Berechtigungen aber nicht wider
5. Das System erneuert den Zwischenspeicher
6. Der Zwischenspeicher spiegelt nun die neuen Berechtiungen wider
7. Der Prozess ist erfolgreich

Durch die Implementation des Mapping-Zwischenspeichers, wird die Ausführzeit von 17 Sekunden auf acht Sekunden reduziert.

Eintrags-Zwischenspeicher

Wird ein Eintrag von 1P angefragt, so soll das Filtermodul diessen lokal zwischengespeichern. Das stellt ein Problem dar, sobald der Eintrag im Entwickler*innen-Vault manuell verändert wird. Das ist jedoch nicht angedacht. Die Entwickler*innen sollten keine Schreibberechtigung auf ihren Vault haben. Werden originale Einträge verändert, werden die Referenzeinträge vom Synchronisierungswerkzeug gelöscht und neu erstellt. Damit erhalten diese eine neue UUID und sind somit im Zwischenspeicher nicht mehr repräsentiert. Durch das Implementieren des Eintrags-Zwischenspeichers wird die Ausführzeit von acht Sekunden auf zwei Sekunden reduziert.

Aktiviertes Caching-Level	**Ausführzeit (Sekunden)**
Ohne Cache	17
UUID-Mapping-Cache	8
Mapping+Item-Cache	2

Tabelle 2.: Ausführzeiten des Test-Playbooks mit Testdaten (Fünf Secrets aus einem 1P-Eintrag)

Sicherheit

Um zu vermeiden, dass sensible Daten, wie zum Beispiel die Secrets in den Eintrags-Daten des Passwortmanagers im Klartext im Zwischenspeicher, für viele Prozesse lesbar, gespeichert werden, werden diese in der Schreib- und Lesefunktion mit AES-256 verschlüsselt. Die Wahl fällt auf AES-256, weil es keine bekannten Angriffe gegen AES-256 gibt, die wesentlich effektiver sind als generische Angriffe gegen Blockchiffren. [BSI , 2024]

5. Evaluation und Fazit

Das abschließende Ergebnis des in dieser Ausarbeitung dokumentierten Projektes ist ein Python-Projekt, das in der Lage ist, anhand einer Berechtigungs-Konfigurationsdatei im Yaml-Format 1P-Einträge in Entwickler*innen-Vaults zu kopieren, zu löschen und deren Referenz zu den originalen Einträgen zu wahren. Diese Berechtigungs-Konfigurationsdatei unterstützt das Erfassen von Berechtigungen über Wildcard-Matching sowie über einzelne UUID-Zuweisungen. Diese Einträge können Entwickler*innen anschließend in der 1P-GUI-Anwendung einsehen.
Das Synchronisations-Werkzeug verweigert Anfragen, die zur Löschung von originalen Einträgen führen könnte.

Ebenso ist Teil des Ergebnisses ein Ansible-Filtermodul, das in der Lage ist, unsensible 1P-Referenzen in Host-Konfigurationsdateien aus 1P zu dereferenzieren, sowohl als interne*r Entwickler*in ohne Entwickler*innen-Vault, als auch als externe*r Entwickler*in mit Entwickler*innen-Vault. Das geschiet in einer IT-sicherheitstechnich unbedenklichen und performanten Art- und Weise.

Hiermit ist das in *⟨⟨1 Lastenheft⟩⟩* definierte Lastenheft des Stakeholders in Gänze erfüllt. Nach Vorstellung der erarbeiteten Ergebnisse vor dem Stakeholder zeigt sich sich dieser zufrieden und bestätigt die pragmatische Korrektheit des Produktes.

5.1. Mehrwert für das Partnerunternehmen

Die hier geschaffene Technologie bringt dem Partnerunternehmen zweierlei Mehrwert: Einerseits können 1P-Berechtigungen nun ohne weiteres beschränkt an Praktikant*innen, neu einngestellte und externe Entwickler*innen vergeben werden. Darüber hinaus bereitet diese Technologie

den Weg für das Partnerunternehmen, ihr Docker-Ansible-Repository so anzupassen, dass es keine Klartext-Secrets mehr beinhaltet. Das erlaubt dem Partnerunternehmen diese Toolbox mit externen Entwickler*innen und Agenturen zu teilen und somit Arbeit auszulagern.

6. Ausblick

Auf diese Umsetzung aufbauend sollten die bestehenden Ansible-Roles im Docker-Ansible-Repository des Partneruntehmens so angepasst werden, dass diese das *resolve_1p_secret* Filtermodul verwenden. Ebenso sollte das bestehende Inventar an Host-Konfigurationsdateien migriert werden, sodass diese keine Klartext-Secrets mehr beinhalten, sondern diese nach 1P verschoben werden und die Host-Konfigurationsdateien lediglich eine IT-sicherheitstechnisch unbedenkliche Referenz auf 1P-Einträge aufweisen.

Literaturverzeichnis

[BSI , 2024] BSI (2024). Cryptographic Mechanisms: Recommendations and Key Lengths. Technical Guideline TR-02102 TR-02102-1, Federal Office for Information Security (BSI). Provides an assessment of the security and long-term orientation for selected cryptographic mechanisms.

[1Password, 2025a] 1Password (2025a). 1Password Secrets Automation . `https://developer.1password.com/docs/secrets-automation/`. Zugriff: Februar 2025.

[1Password, 2025b] 1Password (2025b). Cache item and vault information . `https://developer.1password.com/docs/cli/reference/#cache-item-and-vault-information`. Zugriff: Februar 2025.

[Red Hat, Inc., 2019] Red Hat, Inc. (2019). Ansible module development: getting started . `https://cn-ansibledoc.readthedocs.io/zh-cn/latest/dev_guide/developing_modules_general.html`. Zugriff: Januar 2025.

[Red Hat, Inc., 2025a] Red Hat, Inc. (2025a). Ansible Collaborative - What is Ansible? . `https://www.redhat.com/en/ansible-collaborative`. Zugriff: Januar 2025.

[Red Hat, Inc., 2025b] Red Hat, Inc. (2025b). Filter plugins . `https://docs.ansible.com/ansible/latest/plugins/filter.html`. Zugriff: Februar 2025.

Anhang

A. Stakeholder-Interview

Anforderungserfassung 1Password-Berechtigungsverwaltung und -Schnittstelle
16.10.2024 und 23.10.2024, Bad Dürkheim
Teilnehmer: Leon Etienne, Jochen Stange

Frage: Was sind die Aufgabenbereiche von externen Entwickler*innen?

> Externe steigen z.B. für die Umsetzung großer Projekte ein und arbeiten in diesem Umfang mit uns in agilen Entwicklungsprozessen wie z.B. Scrum. Je nach dem Projekt braucht man da schon einige Zugänge. Es kommt aber auch vor, dass Entwickler wie *<Name geschwärzt>* selbstständig Projekte umsetzen, gegebenfalls auch kleine Projekte. Die brauchen dann eigentlich nur den Backend-Login. Den können wir auch auch mailen. Es war aber auch schon im Diskurs TYPO3-Upgrades outzusourcen. Das ist sowieso undankbare Arbeit und wir wären froh, das aus den Füßen zu haben. Da müssten wir schon einige Zugänge übermitteln.

Frage: Und wieso lagern wir solche Upgrades nicht schon aus?

> Naja die dafür zuständigen Dienstleister brauchen dafür unser Docker-Ansible. Das können wir aber nicht rausgeben, das ist ja voll mit Email und Datenbankzugängen. Wir sind ja schon lange dran das zu fixen, nur wir kommen wir nie so wirklich dazu.

Frage: Könntest du dir vorstellen, diese Zugänge in unseren Passwortmanager auszulagern? Damit könnten wir die Berechtigungsverwaltung an 1Password abgeben.

> Sicher, wenn Ansible dann noch drankommt, klingt das gangbar. Wie funktioniert die Berechtigungsverwaltung da überhaupt?

<Es wird gemeinsam durch die administrative 1Password-Oberfläche gestöbert>

Mh. Ich finde nur Berechtigungen, um Vaults freizugeben, nicht aber für einzelne Einträge. Wir haben doch eigentlich durch unseren relativ teuren Tarif Zugang zu Live-Support. Ich stelle nachher mal eine Support-Anfrage. Die sollen mir erklären, wie das geht. Muss ja gehen.

<Das Interview wird bis nach dem Support-Zoom-Call mit 1Password pausiert>

Leon, das glaubst du nicht. Das geht wirklich nicht. Man kann nur ganze Vaults freigeben. Man kann noch nicht mal Verknüpfungen auf einzelne Einträge in diesen erstellen. Wenn wir jetzt *<Name geschwärzt>* Zugang auf 2 oder 3 Einträge geben wollten, müssten wir die von Hand kopieren und ab dann doppelt Pflegen. Das ist ja unglaublich.

Frage: Ich verstehe. Und wenn wir je Projekt einen Vault erstellen würden?

Dann müssten wir ja den ganzen Passwortmanager umbauen. Außerdem haben wir über 200 Projekte. Mit so vielen Vaults würde sich niemand mehr zurecht finden. Lass das mal lieber sein.

Frage: Ich lasse mir etwas einfallen. Klingt, als müssten wir etwas eigenes bauen. Wäre es OK, wenn wir Zugänge in YAML definieren?

Solange es mit etwas technischem Know-How nicht zu aufwändig zu pflegen ist, gerne. Ich will aber nicht, dass wir nachher für jedes Projekt 20 Einträge einzeln jedem Entwickler zuweisen müssen. Das muss projektbasiert gehen. Trotzdem muss es aber funktionieren, dass wir projektunabhängige Einträge so zuweisen können. Für interne Werkzeuge z.B.

Frage: Vlt. könnten wir Wildcards verwenden, um die Eintragstitel zu durchsuchen? Die sind bei uns ja sehr einheitlich.

> Klingt pragmatisch. Da kann man dann notfalls auch komplette Eintragstitel reinschreiben, oder? Die Einträge, auf die so eine Wildcard passt, werden dann der Entwicklerin zugewiesen.

Frage: Ja, das würde gehen. Vlt wären aber 1Password-IDs eindeutiger. Das wäre auch kein nennenswerter Mehraufwand

> Ok, dann nimm das noch mit dazu. Aber nicht anstatt. Äh und noch was. Das ist ja nicht nur für Ansible. Wie würden damit ja auch Accountzugänge verteilen. Wie sollen die Entwickler das dann einsehen? Über das Terminal wäre das sehr unhandlich.

Frage: Wenn wir Entwicklern eigene Vaults geben würden, in die Einträge hinenkopiert werden, könnten sie die Einträge in der 1P App einsehen.

> Klingt gut. Aber müssen wir die dann nicht doch wieder doppelt pflegen?

Frage: Nein, ich würde einen Mechanismus bauen, der sich darum kümmert, diese Vaults aktuell zu halten. Wenn überhaupt müssen wir einen Sync-Prozess anstoßen. Passt das?

> Wenn du das so hinkriegst, wäre das gut. Zwar nicht ideal, aber 1P scheint ja nicht mehr herzugeben.

Frage: Super. Wenn du sonst keine Fragen oder Anregungen mehr hast, würde ich gleich damit loslegen :)

> Im Moment nicht. Ich melde mich wenn doch.

B. Ideensammlung

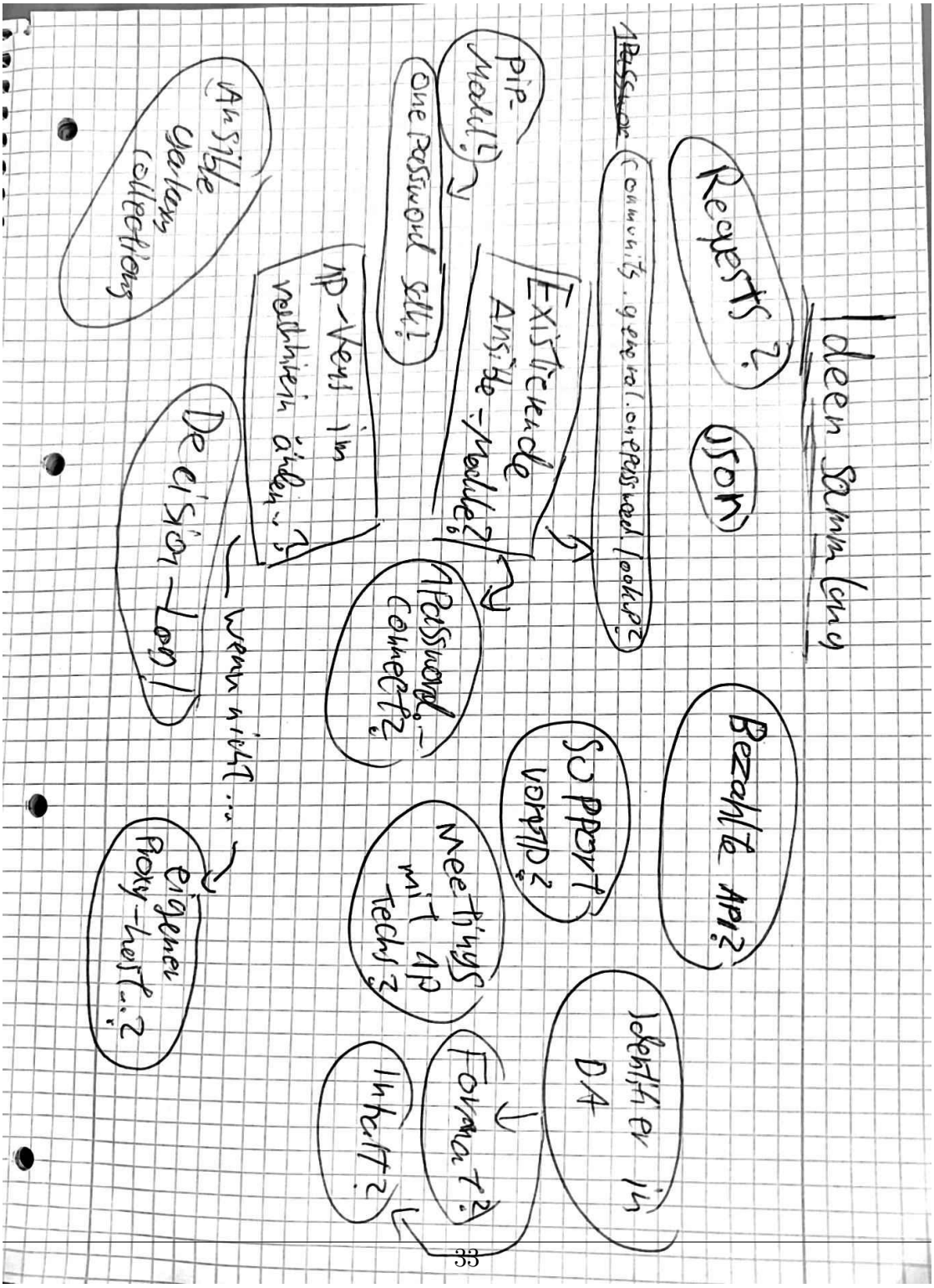


Abbildung 10.: Ideensammlung

Quelle: Eigene Darstellung

C. Relationsdiagramm (Überholt)

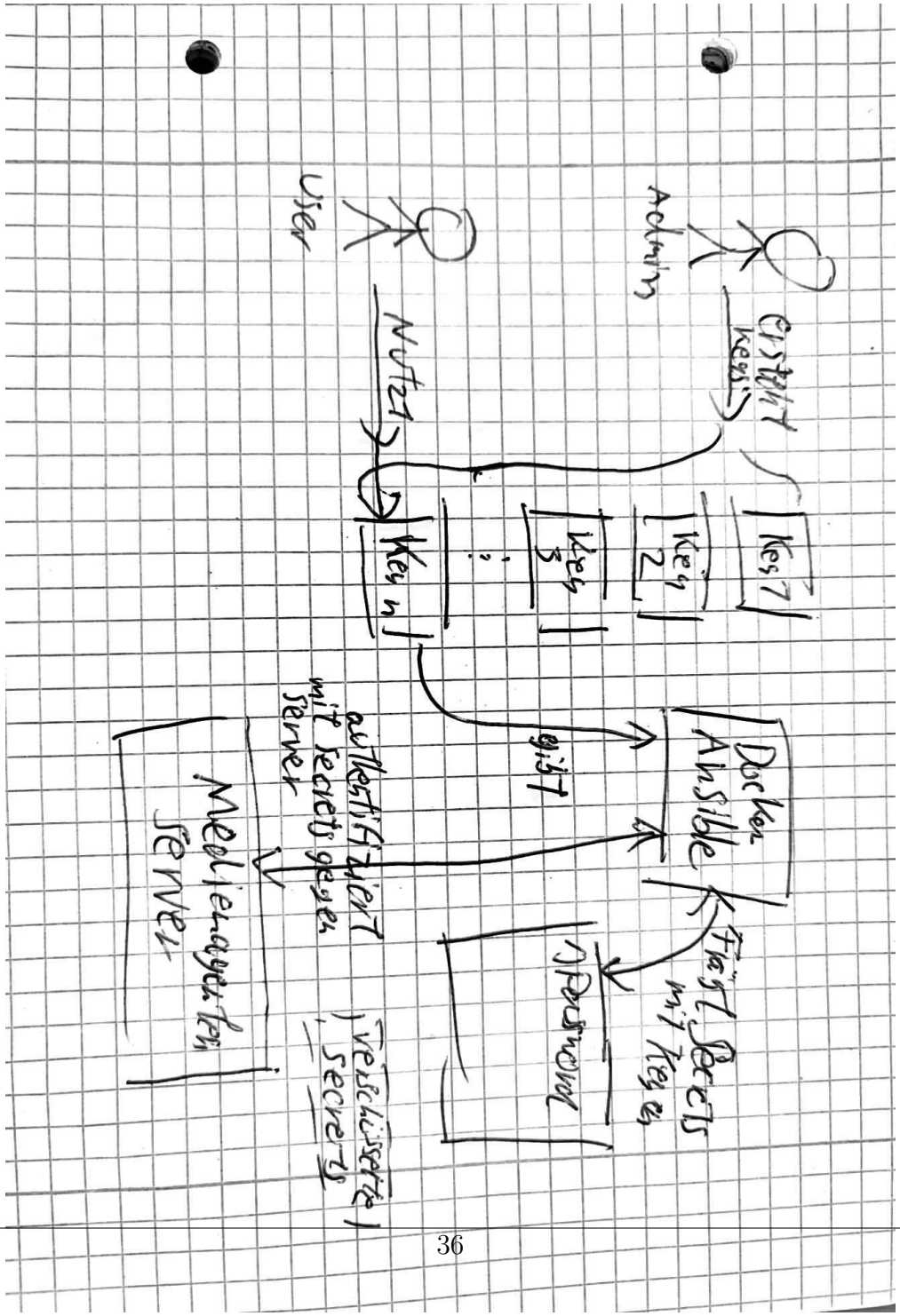


Abbildung 11.: Relationsdiagramm: (Überholt) Relationsdiagramm

Quelle: Eigene Darstellung