# Single Sign-on mit Keycloak – ein Mini-Crashkurs

Univention GmbH

Silke Meyer
silke.meyer@univention.de
@smeyer@univention.social

univention

# Agenda

1. Grundlagen des Web Single Sign-on
2. Das Protokoll OpenID Connect 1.0
3. Einführung in den Keycloak Identity Provider
4. Authentication Flows in Keycloak
5. Absicherung der Keycloak-Installation
6. Mehr-Faktor-Authentifizierung am IdP
7. Debugging

# Grundlagen des Web Single Sign-on

# Was ist Web Single Sign-on?

- „Web“ → Anwendungen, auf die im Browser zugegriffen wird (versus Kerberos)
- Nutzer*innen-Perspektive:
  - Nutzung einer einzigen Browser-Session für den Zugriff auf diverse Anwendungen
  - weniger zu verwaltende Zugangsdaten
- Administrator*innen-Perspektive:
  - keine dienstspezifischen Credentials, Passwörter bleiben immer im Verzeichnisdienst / IdM
  - Absicherung *eines* zentralen Logins (z.B. mit MFA)
  - Kontrolle über Attributfreigaben

# Beteiligte Komponenten

| Identity Provider (IdP / OP) | Service Provider / Relying Party (SP / RP) |
|---|---|
| Authentisierung von Nutzer*innen | Schutz einer Ressource / Anwendung |
| Attributfreigabe nach konfigurierten Regeln | Entgegennahme von Attributen und Weitergabe an die eigentliche Anwendung |
| Übertragung an die Relying Party | Autorisierung nach konfigurierten Regeln (ja/nein, GBAC, RBAC, ABAC, ...) |

# Ablauf im einfachsten Fall

- Nutzer*in klickt im Browser bei Dienst „Login“
- Umleitung zur Loginseite eines IdP
- Login → IdP authentisiert → Authentication
- Umleitung zur geschützten Anwendung → IdP überträgt Informationen an SP
- SP prüft, ob aufgrund der Infos Zugriff gegeben wird → Authorization
- Nutzer*in ist eingeloggt und darf Dinge tun
- Nutzer*in klickt auf zweiten angeschlossenen Dienst und ist schon angemeldet

# Optionale Komponenten

- Discovery Service / Broker
  - Vermittlungsstelle
  - Auswahl eines IdP aus einer Liste und Umleitung zu richtigen IdP
  - bei Diensten, für deren Nutzung man sich mit mehr als einem IdP anmelden kann
- Proxy
  - Übersetzung zwischen Protokollen (SAML → OIDC, OIDC → SAML, SAML → SAML mit Umschreibung)
  - Verbindung verschiedener Identity Provider oder ganzer Identity-Föderationen

# Vertrauensstellung

- Wie heißt die Gegenstelle? Mit welchen Zertifikaten kann die signierte Kommunikation validiert werden? Welche Kommunikationsstandards unterstützt die Gegenstelle? An welche Endpunkte sollen die Requests geschickt werden?
- Vorab-Austausch bestimmter Eckdaten für die Kommunikation: Metadaten
  - eindeutige Identifier der Systeme
  - Kommunikationsendpunkte
  - Zertifikate für Signierung / Verschlüsselung
- OIDC: Standard-Endpunkt für OIDC-Metadaten unter .well-known/openid-configuration

# Das Protokoll OIDC 1.0

# Das Protokoll OIDC 1.0

| OIDC 1.0 (2014) |
| --- |
| Spezifikation durch OpenID Foundation |
| Erweiterung von OAuth 2.0 um Konzept von Identität (ID Token) |
| Browser, native Apps, Geräte (versch. Flows) |
| JSON-basiert, HTTP-/REST-API-Kommunikation |
| geringe Payload |
| Vertrauen über Vorab-Austausch von Client ID, Client Secret u. Redirect URI |

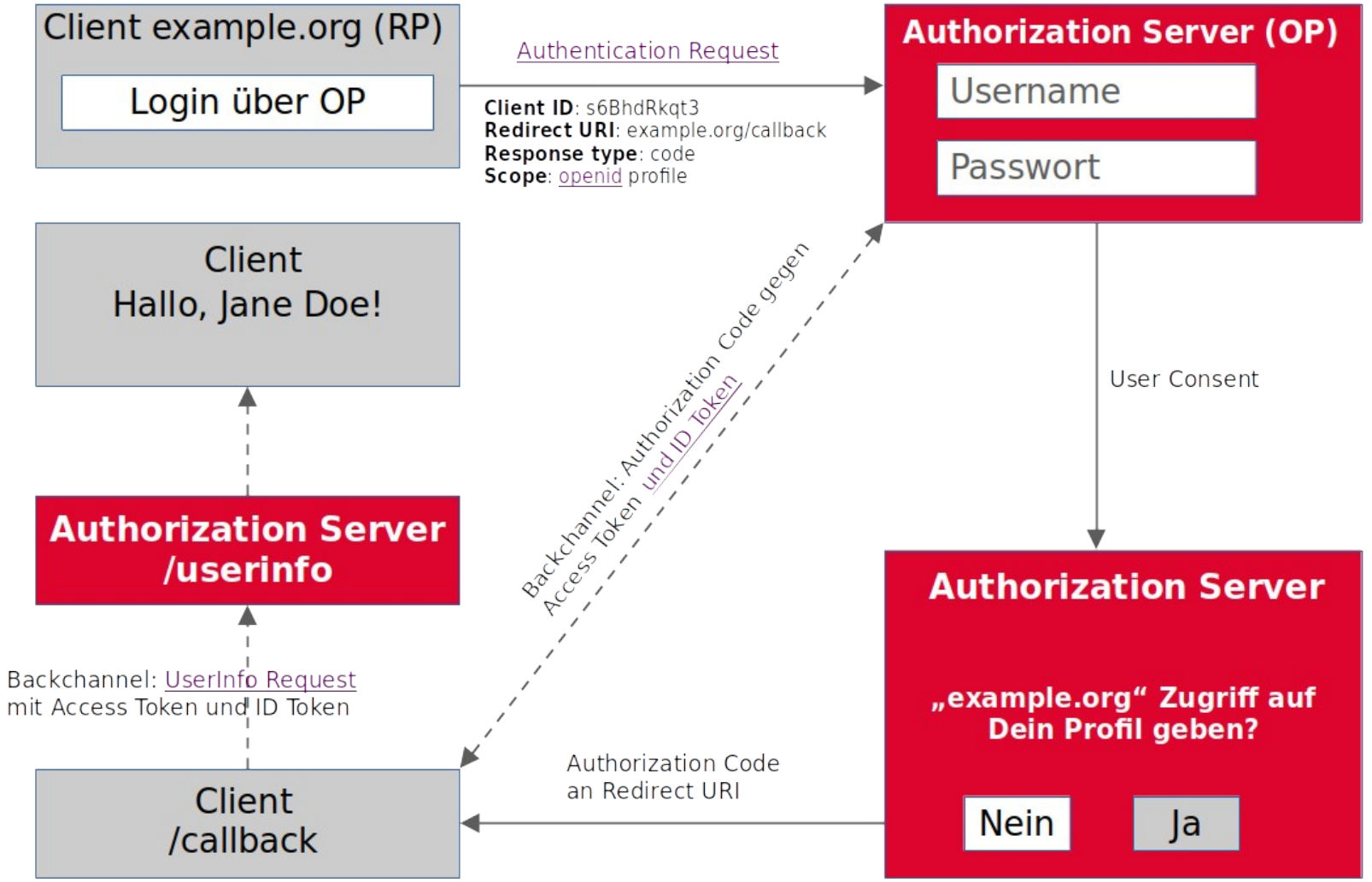
# Terminologie (Auszug)

| OIDC | |
|---|---|
| OpenID Connect Provider / OP | der Identity Provider, z.B. Keycloak |
| Relying Party / RP | angebundener Dienst, der die Authentisierung an den OP delegiert |
| Client ID | eindeutige Kennung einer RP für die Kommunikation mit einem OP |
| Claim | Attribut (am Nutzerkonto) |
| Scope | thematisch gebündelte Claims |
| Access Token, Refresh Token, ID Token (JWT) | die Tokens, die der OP für den User ausstellt und der RP schickt |
| Signieren/Verschlüsseln mit JWK | |

# Von OAuth 2.0 (2012) zu OIDC 1.0 (2014)

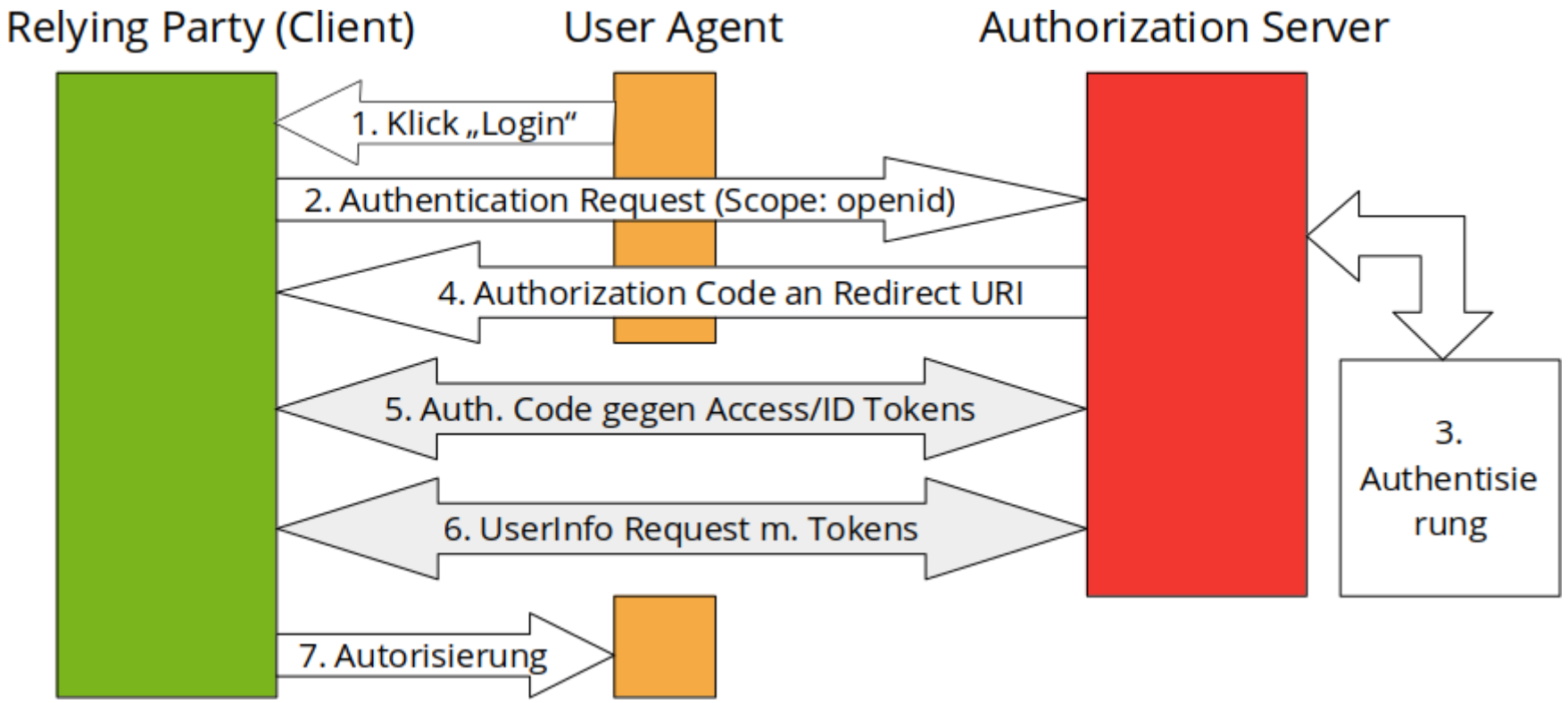
- OAuth 2.0
  - Informationen mit Dritten teilen (z.B. mit einer Website)
  - delegierte Autorisierung ohne Preisgabe des Passwortes
  - auch API-Autorisierung über Access Tokens (JSON Web Tokens)
- OIDC 1.0
  - Erweiterung von OAuth 2.0 um einen ID Token zur Abbildung einer Identität
  - ID Token enthält einen i.d.R. generierten Identifier, den subject claim
    - subject public: IdP generiert pro Nutzerkonto einen Identifier
    - subject pairwise: IdP generiert pro Nutzerkonto pro Dienst einen Identifier
  - angefragter zusätzlicher Scope „openid“ im Authentication Request löst den Unterschied zu OAuth 2.0 aus

# OIDC 1.0 (Authorization Code Flow)




Quelle: Nate Barbettini, Okta https://www.youtube.com/watch?v=996OiexHze0

# OIDC 1.0 (Authorization Code Flow)



# OIDC Token Response (gekürzt)

- Übertragung im HTTP-Header
- HTTP/1.1 200 OK

```
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache
{
"access_token": "SlAV32hkKG",
"token_type": "Bearer",
"refresh_token": "8xLOxBtZp8",
"expires_in": 3600,
"id_token": "eyJ0eXAiOiJKiJIUzI1NiJ9.EyJpc3MiOiJA4MTkzOD.dBjft4CVP-mhb1p1r_wWOEjXk"
}
```

# Einführung in den Keycloak Identity Provider

# Allgemeines zum Keycloak IdP

- Entwicklung unter dem Dach von IBM/RedHat (Apache 2.0 Lizenz)
- Java + Datenbank für persistente Daten (Default: Postgresql)
- Anbindung an versch. Verzeichnisdienste und/oder lokale User-Datenbank
- SAML 2.0, OAuth 2.0, OIDC 1.0
- Konfiguration über Webinterface oder REST API
- Mandantenfähigkeit (separate „Realms“ möglich)
- ausgelegt auf Betrieb in Cloud-Infrastruktur: Skalierbarkeit, Clusterfähigkeit

# Betriebsmodi und Grundkonfiguration

- entpacktes Archiv, Docker, Podman, K8s, OpenShift
- bei Betrieb im Docker-Container: docker-compose.yml mit Parametern für den Java-Prozess im Container
  - → siehe Beispiel
  - wichtig:  Loglevel wird als Startparameter mitgegeben → keine Änderung des Loglevels im laufenden Betrieb!
- bei Betrieb von lokal entpacktem Archiv: Konfigurationsdatei für den Start des Java-Prozesses, eigenes systemd unit file (u.a. Nachteile manuell entpackter Archive)

# Konfiguration „von innen“

- Konfiguration der eigentlichen Inhalte innerhalb von Keycloak über GUI oder API
  - Realm(s)
  - Im Realm: Backendkonfigurationen, angebundene Clients, Attributmappings, Regeln zur Attributfreigabe, ggf. User Consent, Events, wenn gewünscht lokale Nutzerkonten, uvm.
- Speicherung in Keycloak-Datenbank → Zugriff von mehreren Instanzen möglich
- Import- und Export-Möglichkeit

# Die Weboberfläche von Keycloak

- Auswahl des gewünschten Realms (Mandant)
- unter Realm-Einstellungen:
  - URLs zum Abruf der Metadaten des OpenID Connect Provider und des SAML IdPs
  - Anschalten des Event-Logs → Speicherung in Datenbank
- unter User Federation
  - Anbindung des Verzeichnisdienstes
  - Konfiguration / Mapping von Attributen
- unter Sessions: Übersicht über aktive IdP-Sessions
- unter Ereignisse: Übersicht über aktuelle Login-Vorgänge

# Die Weboberfläche von Keycloak

- unter Clients: Übersicht der angebundenen Dienste
  - Zugang zu den Einzelkonfigurationen
  - Import und Export von Client-Konfigurationen
  - Vorschau auf Token-Inhalte → Debugging
- unter Client Scopes: Sammlungen von Attributen/Claims und Angaben darüber, wie mit dem jeweiligen Scope *standardmäßig* umgegangen werden soll.
  - gezielte Freigabe von Scopes in einzelnen Client-Konfigurationen
  - dort auch: Unterreiter „Evaluate“ / „Auswerten“: Vorschau auf die Tokens für einen bestimmten User, der sich für einen bestimmten Client anmeldet

# Infinispan: Der Session Cache

- Cluster zwischen mehreren Keycloak-Instanzen möglich
- Quorums-Logik
- Sessions werden auf dem Node vorgehalten, auf dem sie gestartet wurde
- Weitergabe an die anderen Nodes auf Anfrage

# Authentication Flows in Keycloak

# Authentication Flows

- Abfolge / Gruppierung von Schritten, die Keycloak bei der Authentisierung von Nutzer*innen (oder Clients) ausführt
- *nicht zu verwechseln* mit den Authentifizierungsmethoden, die in OIDC 1.0 spezifiziert sind (Authorization Code Flow, Implicit Flow etc.)
- gängige Authentication Flows werden von Keycloak mitgeliefert
- Auth Flows sind immer einem Realm zugeordnet.
- Auth Flows können kopiert, abgewandelt oder von Grund auf neu angelegt werden.
- hierarchischer Aufbau mit erforderlichen und optionalen Schritten

# Vorkonfigurierte Authentication Flows in Keycloak

- Browser Flow
  - Endnutzer*innen, die mit Browser auf Anwendungen zugreifen
  - hier im Fokus / von unseren Kund*innen bisher mit Abstand am meisten genutzt
- First Broker Login
  - Umgang mit erstmaligen Logins von anderen Identity Providern (IdP)
  - für föderierte IdPs

# Der Browser Flow

- Prüfung auf gültige Session (Cookie, Kerberos-Ticket im Browser)
- Prüfung, ob Kriterien für die Umleitung zu einem anderen IdP erfüllt sind
- optionaler Sub-Flow „Organization“ (ab Keycloak 26)
- Sub-Flow „Forms“: der wesentliche Teil
- Username und Passwort müssen angegeben werden
- 2FA mit OTP ist optional, wird nur abgefragt, wenn es im Nutzerkonto konfiguriert ist

# First Broker Login

- für Föderation mit anderem IdP
- Schritte, die bei Anmeldung über einen anderen IdP erfolgen sollen
- Prüfung des eigenen Profils durch Nutzer*in
- Anlegen oder Verlinken des Nutzerkontos (in der Keycloak-Datenbank)
- (ggf. Aufnahme in Organisation)
- Konfigurierbar ist, ob
  - dieser oder ein eigener Flow benutzt werden soll,
  - nach dem Login noch ein Flow durchlaufen werden soll

# Use Cases für verschiedene Auth Flows

- 2FA für einen bestimmten Dienst erzwingen
- 2FA nur für manche Rollen erzwingen
- 2FA für manche Rollen erzwingen, für alle anderen optional anbieten
- Föderation mit anderem Identity Provider
- (Einbindung von Faktoren aus einem extern verwalteten Backend)

# Eigene Auth Flows definieren

- Lektüre der Bespiele in der Keycloak-Dokumentation
- Start mit Vorlagen / bestehenden Flow duplizieren und abwandeln
- Bedingungen nur innerhalb von „Conditional Sub-Flows“ nutzbar
- Auswahl zeigt alle möglichen Bedingungen an, Negation ist möglich
- Verbindung mit expliziten Deny-/Allow-Direktiven möglich
- Ändern der Reihenfolge mit Drag and Drop mehr oder weniger möglich ;)
- Zuordnung von Auth Flows zu Clients in erweiterten Client-Einstellungen
  - Ersetzen des globalen Standard-Browser-Flows im Realm durch eigenen Flow
  - Zuweisen eines eigenen Flows zu einer bestimmten Client-Anwendung (in den erweiterten Client-Einstellungen)

# Tipps zum Erstellen eigener Auth Flows

- Keep it simple!
- bestehende Flows erst verstehen, dann abwandeln
- mitgelieferte Flows können durch Updates verändert werden!
- das Verhalten eigener Flows testen:
  - mit verschiedenen Nutzerkonten bzw. Rollen
  - beim Zugriff auf verschiedene Clients in verschiedenen Reihenfolgen
  - beim initialen Login / mit valider IdP-Session / beim Auffrischen einer laufenden Session

    → auf nicht intendierte Nebeneffekte prüfen

# Absicherung der Keycloak-Installation

# Keycloak absichern

- Abschnitt „Mitigating Security Threats“ in der Keycloak-Dokumentation, Buch *„Keycloak – Identity and Access Management for Modern Applications“*
- Software aktuell halten, keine LTS-Version bei frei verfügbarer Variante
- Erreichbarkeit für Clients auf Port (80 und) 443 (TLS 1.3)
- bei Loadbalancing mit SSL-Terminierung: Neuverschlüsselung des Traffics LB → KC
- Admin-Konsole und REST-API nur aus internen Netzen erreichbar machen

# Keycloak absichern

- Keycloaks Hostname in Startparameter oder Realm-Einstellungen explizit setzen
  - Keycloak verwendet sonst Host Header aus eingehenden Requests
  - Angriffsvektor bei Links zum Passwort-Reset
- Best Practice: regelmäßige Rotation des Schlüsselmaterials: Realm-Einstellungen -> Keys
  - Abwägung: Rotation nicht kompromittierter Schlüssel bei schwer umzusetzender Automatisierung?
- Brute Force-Schutz für User-Logins aktivieren: Realm-Einstellungen -> Security Defenses -> Reiter Brute Force Detection
- ggf. in Verbindung mit fail2ban

# Keycloak-Absicherung über Firewall / WAF

- Absicherung bestimmter Endpunkte
  - Keycloak Admin Console
  - REST API
- Notwendig sind:
  - eingehender Traffic auf Port 443
  - ausgehender Traffic zum Identity Management System (z.B. OpenLDAP)
  - ausgehender Traffics zur Keycloak-Datenbank
  - ausgehende Requests über HTTPS zu angebundenen Anwendungen (z.B. für Logout Requests) oder zu anderen Identity Providern (z.B. für Token Requests)

# Keycloak-Datenbank

- enthält die gesamte IdP-Konfiguration inkl. angebundenen Anwendungen
- bei lokalen Nutzerkonten: Konten und gehashte Passwörter
- ggf. LDAP Bind-Credentials, private Schlüssel für Signaturen, SMTP-Credentials

  → hierfür bei entsprechendem Schutzbedarf: Vault verwenden
- Firewall vor die DB ;)
- nur authentifizierte Zugriffe auf DB erlauben, wenn über Netzwerk, dann über TLS
- Datenbank-Backups an einem sicheren Ort aufheben

# Infinispan: Keycloaks Session Cache

- enthält keine Informationen über die Nutzerkonten oder Signing Keys, nur Sessions
- keine Verwendung der Session-Information ohne signierten Token oder Cookie
- Cluster-Traffic kann, wenn gewünscht, verschlüsselt werden
- verschiedene Wege → der einfachste mit symmetrischer Verschlüsselung
  - Shared Secret im Java Keystore auf jedem Node ablegen
  - Cache-Konfiguration conf/cache-ispn.xml bearbeiten (Beispiel für verschlüsselte udp-Kommunikation in Keycloak-Doku)

# Mehr-Faktor-Authentifizierung mit Keycloak

univention

# Absicherung der Nutzerkonten

- Passwort-Richtlinien (Authentication → Policies für lokale Nutzerkonten)
- Brute-Force-Schutz (Bordmittel und/oder fail2ban)
- Datensparsamkeit:
  - Beschränkung auf das Vorhalten der nötigsten Attribute in Keycloak
  - feingranulare Regeln zu Herausgabe von Attributen an angebundene Dienste, ggf. eigene Scopes[1] definieren
- Zwei- oder Mehrfaktor-Authentifizierung am Identity Provider

[1] In OIDC ist ein Scope ein „thematisch gebündeltes Päckchen“, das mehrere Nutzerattribute enthalten kann.

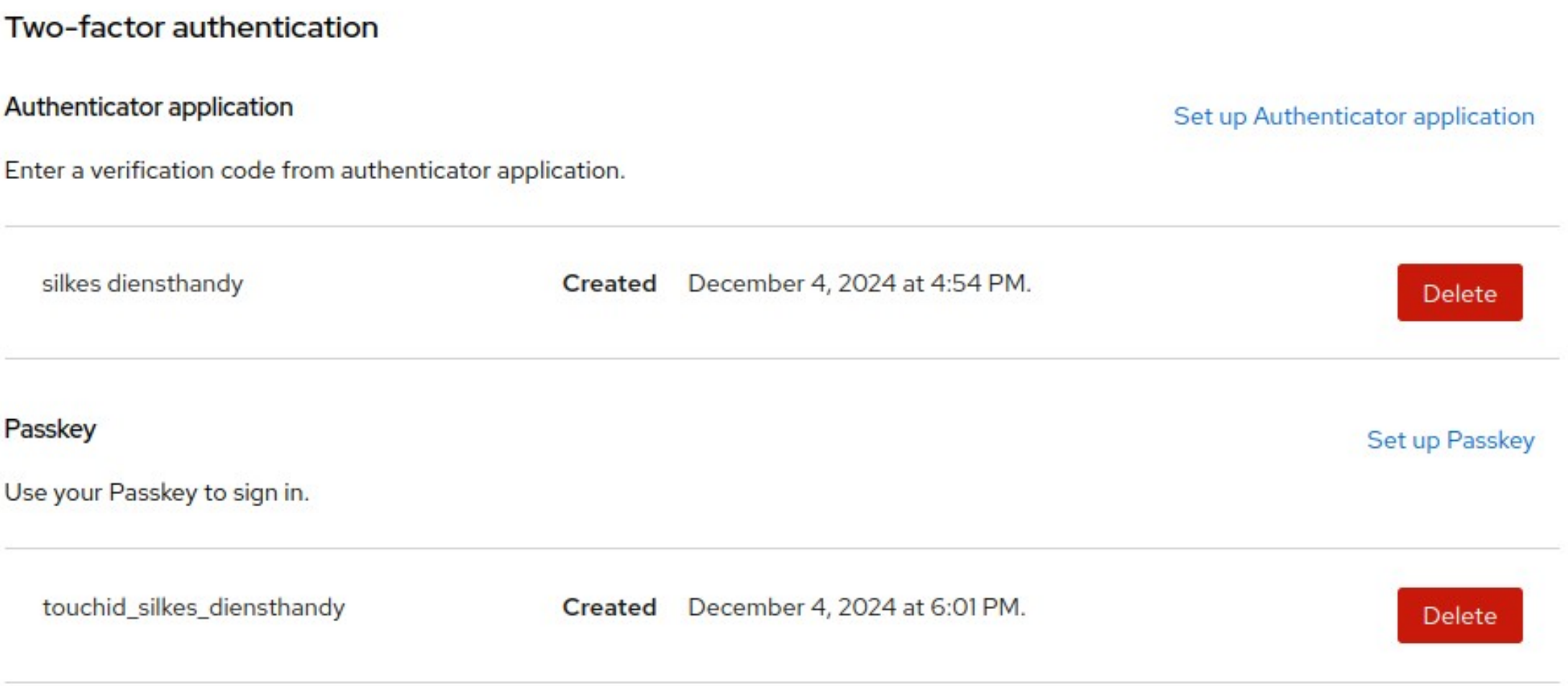
# Mehr-Faktor-Authentifizierung am IdP

- IdP als potenzielles zentrales Angriffsziel schützen
- Schutzbedarf evaluieren
- Design-Entscheidung zur Token-Verwaltung
  - Wo / durch wen werden Tokens administriert?
  - Selfservice bei Wiederherstellung / Geräteverlust
  - Account Management durch die Nutzer*innen unter `https://login.example.org/realms/myrealm/account`
- hier Fokus auf Keycloaks mitgelieferte Möglichkeiten (ohne 3rd-Party-Plugins)

# MFA: Überblick über mögliche Faktoren

- offizielle Keycloak-Dokumentation zu MFA
- hier keine Berücksichtigung von Plugins
- Einmalpasswörter mit Smartphone-App (TOTP)
- Webauthn / FIDO 2 (oder Vorgänger U2F) mit USB-Tokens oder Lesegeräten für biometrische Credentials
- x509-Zertifikate (siehe Keycloak-Dokumentation)
- Einrichtung eines zweiten Faktors beim nächsten Login verpflichtend machen: möglich über Gestaltung des Authentication Flows
- Recovery Codes zur Nutzung bei Geräteverlust

# MFA in den Kontoeinstellungen



# Debugging

# SSO-Debugging

- An welcher Stelle im Login-Flow tritt der Fehler auf?
- Welche Seite meldet den Fehler? → beide Seiten betrachten, wenn möglich
- versch. Browser / Profile nutzen (Cache und Cookies sauber)
- Logs
  - OpenLDAP: slapd-Log z.B. auf „stats“ setzen (syslog/Journal)
  - Loglevel Keycloak auf „DEBUG“
  - `docker logs -f <keycloak-container> --since 5m`
- Log der jeweiligen Client-Anwendung prüfen (lassen)
- Token-Auswertungstool in Keycloak verwenden (Client → Client Scopes → Evaluate)

# SSO-Debugging

- Sind die in den JSON-Konfigurationen / xml-Metadaten publizierten Kommunikationsendpunkte erreichbar?
- Stimmt das tatsächlich verwendete Schlüsselmaterial überein mit dem in den JSON-Konfigurationen / xml-Metadaten angegebenen?
- Loadbalancer oder Proxy vor dem IdP? → X-Forwarded-For Header?
- Welche Attribute / Claims erwartet die geschützte Anwendung? Welche werden vom IdP übertragen?

# Quellen und weiterführende Links

- offizielle Keycloak-Dokumentation
- Stian Thorgersen/Pedro Igor Silva:  Keycloak - Identity and Access Management for Modern Applications, 2. Aufl., Packt Publishing, Birmingham/Mumbai 2023
- Informationen zur von Univention mitgelieferten Keycloak-App
- Help-Artikel „Q&A: How to enforce the use of OTP for special clients“ von C. Scheinig (Univention)
- Informationen zum neuen Feature „Organisationen“ ab Keycloak 26: Niko Köbler: Multi Tenancy in 1 Realm (Youtube)
- Vortrag Stefan Schumacher CLT 2025: Passwortlose Logins mit Passkeys, guter Überblick

# Vielen Dank für Ihr Interesse! Fragen?

Silke Meyer
IT Consultant
silke.meyer@univention.de
@smeyer@univention.social
+49 421 22232-106