

Security und Spring Boot

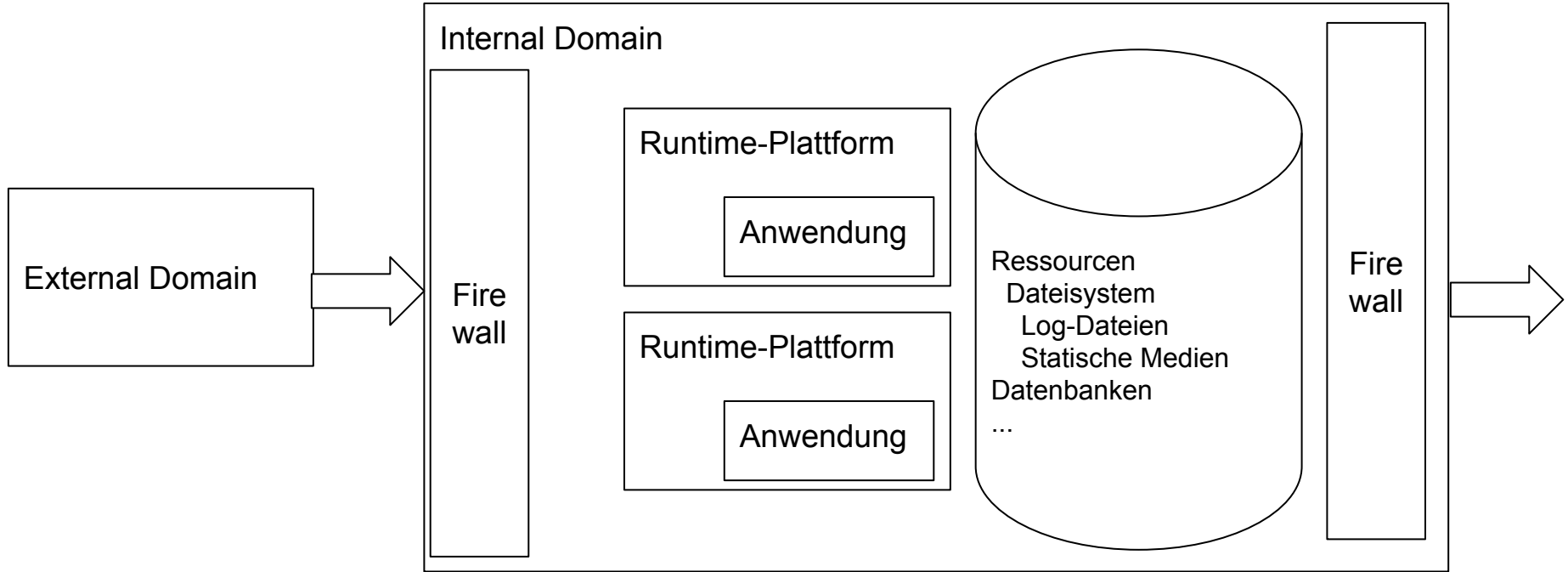
Einführung

Common Sense

- Software-Programme sind anfällig gegen typische Hacker-Angriffe
- Die dadurch hervorgerufenen Schäden können selbst bei trivial scheinenden Lücken immens sein
 - Konkreter Schaden
 - Prestige- und Image-Verlust
- Es ist deshalb obligatorisch, zumindest die typischen Security-Lücken von vornherein auszuschließen und alle anderen durch ständige Qualitätssicherung zu erkennen und zu beheben

- Verschlüsselte Kommunikation
 - https
 - VPN
- Firewall
 - z.B. Denial of Service
 - Port Scans
 - Injection
- Hardening
 - Betriebssystem
 - Insbesondere Einführung von Zugriffsbeschränkungen auf das Dateisystem und Netzwerk
 - Prinzip “Least Privileges”
 - Runtime
 - Konfiguration, z.B. deaktivieren überflüssiger Ports/Protokolle

- Authentifizierung
 - Benutzerverwaltung mit Benutzername/Password, Credentials
 - Directory Server (LDAP)
 - Datenbank
- Autorisierung
 - Anwendung definiert Rollen, denen bestimmte Berechtigungen erteilt werden
- Daten-Validierung bei jeglichem Transfer zwischen Domänen



- Security-API
 - JEE
 - in der Praxis total proprietär
 - Spring Security
 - de facto Standard
- Authentifizierung und Autorisierung
 - Identity Management
 - Rules Engine
 - Schlüssel- und Zertifikatsverwaltung
-

Sichere Kommunikation

- Verfügbarkeit (availability)
- Integrität (integrity)
- Vertraulichkeit (confidentiality)
- Verbindlichkeit (liability)

- Grundidee
 - Mit wenig Aufwand kann eine Information so verschlüsselt werden, dass ein Auslesen ohne Kenntnis eines Secrets nur mit immenser Aufwand möglich ist
- Stand heute
 - Normal-Anwender: Jahrmillionen
 - Organisierter Hacker: Ein paar Jahre
 - Organisierter Hacker mit Backdoors: Echtzeit

- Daten werden unidirektional in einen Hashwert überführt
- “Unendlich” -> Endlich

- Algorithmen sind sehr schnell
- Der Schlüssel muss dem Sender und dem Empfänger bekannt sein

- Schlüssel besteht aus zwei Teilen
 - public Key
 - private Key dient zur eindeutigen Definition einer Identität
- Algorithmen sind relativ langsam, Schlüssel-Breiten sind wesentlich länger als bei der symmetrischen Verschlüsselung

- Generierung
- Zertifizierung
- Verwaltung
- Verifizierung



Bundesamt für Sicherheit in der Informationstechnik

Das BSI Themen Aktuelles Presse Publikationen

IT-Grundschutz-Kataloge

Startseite IT-Grundschutz
Inhalt
• Allgemeines
• Bausteine
• Gefährdungskataloge
• **Maßnahmenkataloge**
 ▶ M 1 Infrastruktur
 ▶ **M 2 Organisation**
 ▶ M 3 Personal
 ▶ M 4 Hardware und Software
 ▶ M 5 Kommunikation
 ▶ M 6 Notfallvorsorge
• Rollendefinitionen
• Glossar
• Index A-Z
Baustein Datenschutz
Hilfsmittel
Überblickspapiere
Bezugsquellen
FAQ
Registrierung / Newsletter
Download
Kontakt

Suche Suchbegriff eing. ▶

» Startseite » Themen » IT-Grundschutz-Kataloge » Inhalt » Maßnahmenkataloge » M 2 Organisation » M 2.46 Geeignetes Schlüsselmanagement

M 2.46 Geeignetes Schlüsselmanagement

Verantwortlich für Initiierung: IT-Sicherheitsbeauftragter

Verantwortlich für Umsetzung: Fachverantwortliche, IT-Sicherheitsbeauftragter

Die Verwendung kryptographischer Sicherheitsmechanismen (z. B. Verschlüsselung, digitale Signatur) setzt die vertrauliche, integre und authentische Erzeugung, Verteilung und Installation von geeigneten Schlüsseln voraus. Schlüssel, die Unbefugten zur Kenntnis gelangt sind, bei der Verteilung verfälscht worden sind oder gar aus unkontrollierter Quelle stammen (dies gilt auch für die Schlüsselvereinbarung zwischen Kommunikationspartnern), können den kryptographischen Sicherheitsmechanismus genauso kompromittieren wie qualitativ schlechte Schlüssel, die auf ungeeignete Weise erzeugt worden sind. Qualitativ gute Schlüssel werden in der Regel unter Verwendung geeigneter Schlüsselgeneratoren erzeugt (siehe unten). Für das Schlüsselmanagement sind folgende Punkte zu beachten:

Schlüsselerzeugung

Die Schlüsselerzeugung sollte in sicherer Umgebung und unter Einsatz geeigneter Schlüsselgeneratoren erfolgen. Kryptographische Schlüssel können zum einen direkt am Einsatzort (und dann meistens durch den Benutzer initiiert) oder zum anderen zentral erzeugt werden. Bei der Erzeugung vor Ort müssen meistens Abstriche an die Sicherheit der Umgebung gemacht werden, bei einer zentralen Schlüsselerzeugung muss sichergestellt sein, dass sie ihre Besitzer authentisch und kompromittierungsfrei erreichen.

Geeignete Schlüsselgeneratoren müssen kontrollierte, statistisch gleichverteilte Zufallsfolgen unter Ausnutzung des gesamten möglichen Schlüsselraums produzieren. Dazu erzeugt z. B. eine Rauschquelle zufällige Bitfolgen, die mit Hilfe einer Logik nachbereitet werden. Anschließend wird unter Verwendung verschiedener Testverfahren die Güte der so gewonnenen Schlüssel überprüft.

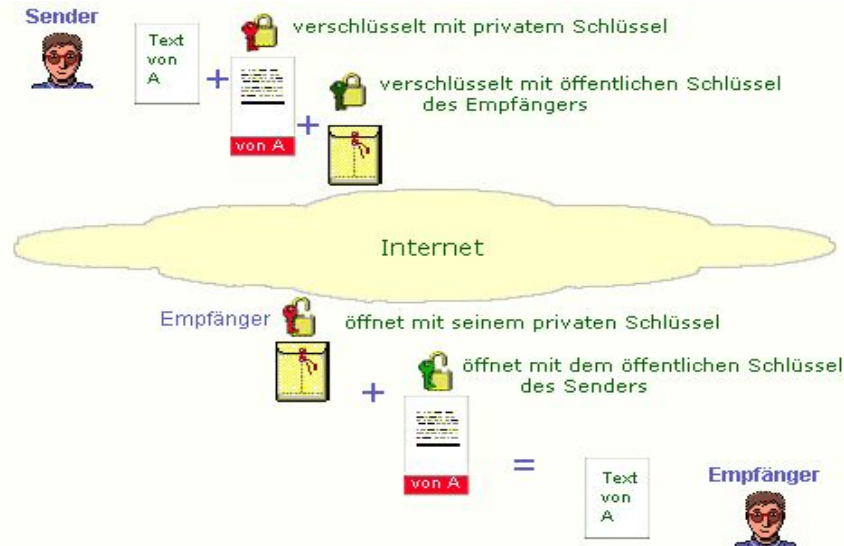
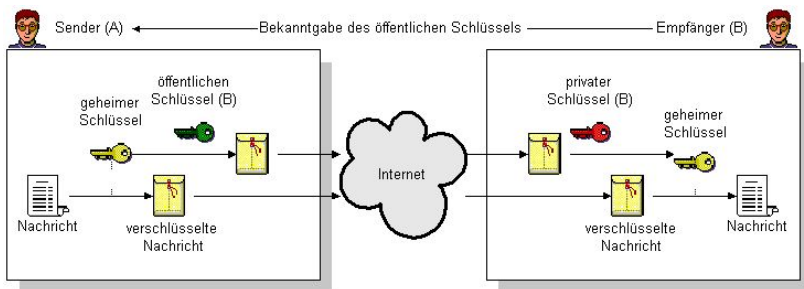
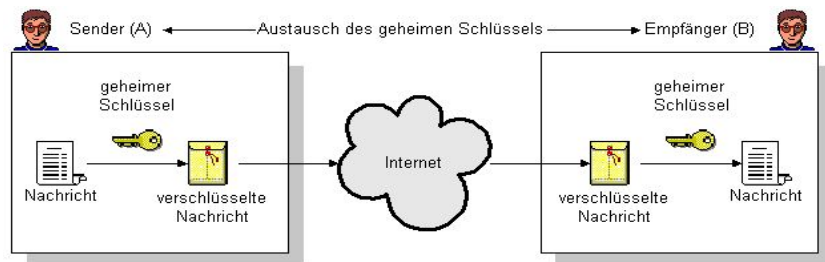
Einige Kryptomodule, insbesondere solche, die keinen integrierten Zufallszahlengenerator besitzen, greifen auf Benutzereingaben zur Schlüsselerzeugung zurück. Beispielsweise werden hier Passwörter abgefragt, aus denen dann ein Schlüssel abgeleitet wird, oder der Benutzer wird gebeten, beliebigen Text einzutippen, um zufällige Startwerte für die Schlüsselgenerierung zu erhalten. Solche Passwörter sollten dabei gut gewählt sein und möglichst lang sein. Wenn möglichst "zufällige" Benutzereingaben angefordert werden, sollten diese auch zufällig, also schlecht vorhersagbar, sein.

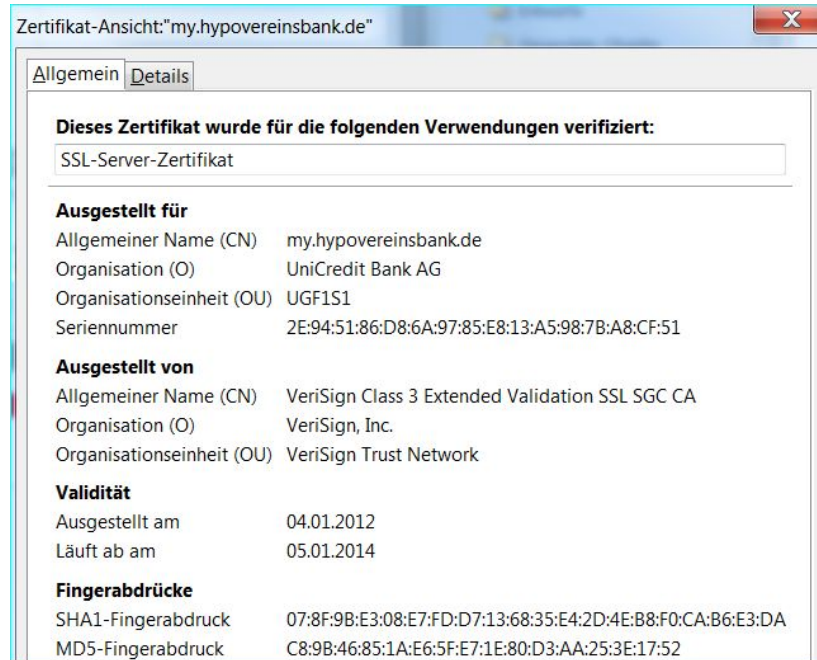
Schlüsseltrennung

Kryptographische Schlüssel sollten möglichst nur für einen Einsatzzweck dienen. Insbesondere sollten für die Verschlüsselung immer andere Schlüssel als für die Signaturbildung benutzt werden. Dies ist sinnvoll,

- damit bei der Offenlegung eines Schlüssels nicht alle Verfahren betroffen sind,
- da es manchmal erforderlich sein kann, Verschlüsselungsschlüssel weiterzugeben (Vertretungsfall),
- da es unterschiedliche Zyklen für den Schlüsselwechsel geben kann.

Schlüsselverteilung / Schlüsselaustausch





Jetzt von
Symantec

VeriSign
Authentifizierungsdienste

Deutschland [ändern] | Kontaktieren Sie uns



powered by VeriSign

Produkte & Services

Partner

Support

Mein Konto


powered by VeriSign

Dasselbe Häkchen. Ein neuer Name.

Derselbe leistungsstarke Schutz.

Dieselbe Qualität bei Sicherheit, Services und Support, der Sie bei VeriSign vertrauen, wird Ihnen nun von Symantec geboten.

Die Konsequenzen für Sie >

1 2 3 4

KAUFEN

SSL-Zertifikate

KAUFEN

Symantec™ Safe Site

KAUFEN

Code-Signing

TESTEN

Kostenlose Demoversion

ERNEUERN

SSL-Zertifikate erneuern

ANMELDEN

Symantec™ Trust Center


powered by VeriSign

Norton™ Secured-Siegel

Verwalten Sie mehrere SSL-Zertifikate?

Eine einfache Management-Lösung.



Weitere Informationen >

Schützen Sie Ihre Website. Steigern Sie Ihren Umsatz.

Neue Funktionen in Symantec SSL machen es einfach, Ihrer Website **zu vertrauen**.



Weitere Informationen >

 VERISIGN™

Für Online-Sicherheit und Verfügbarkeit verlässt sich Ihr Unternehmen auf:

- Managed DNS
- DDoS Protection
- iDefense™
- Domain Name Services

Diese Services sind bei VeriSign Inc unter VerisignInc.com erhältlich.

Kontaktieren Sie uns

Über Symantec

Neuigkeiten

Blogs

Rechtliche Hinweise

Datenschutz

Repositoryum

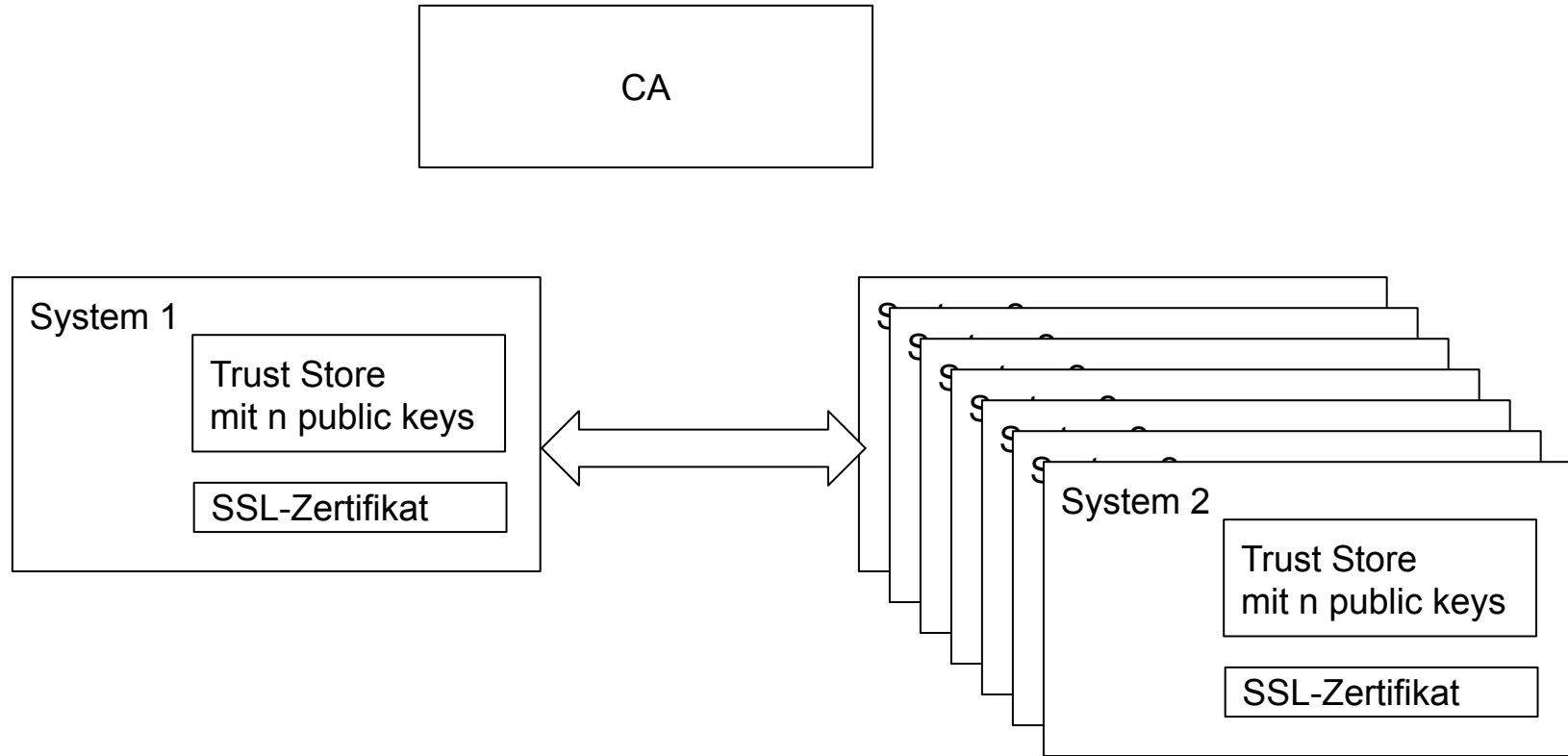
Globale Sites

Sitemap

SSL

- Unsichere Verbindung, http
- Client-Server-Zugriff
 - Austausch und Verifikation der Zertifikate
 - Über eine asymmetrische Verschlüsselung wird eine erste verschlüsselte Kommunikation aufgebaut
 - Über diese wird ein symmetrischer Schlüssel ausgetauscht
 - Über diesen erfolgt der Rest der Kommunikation

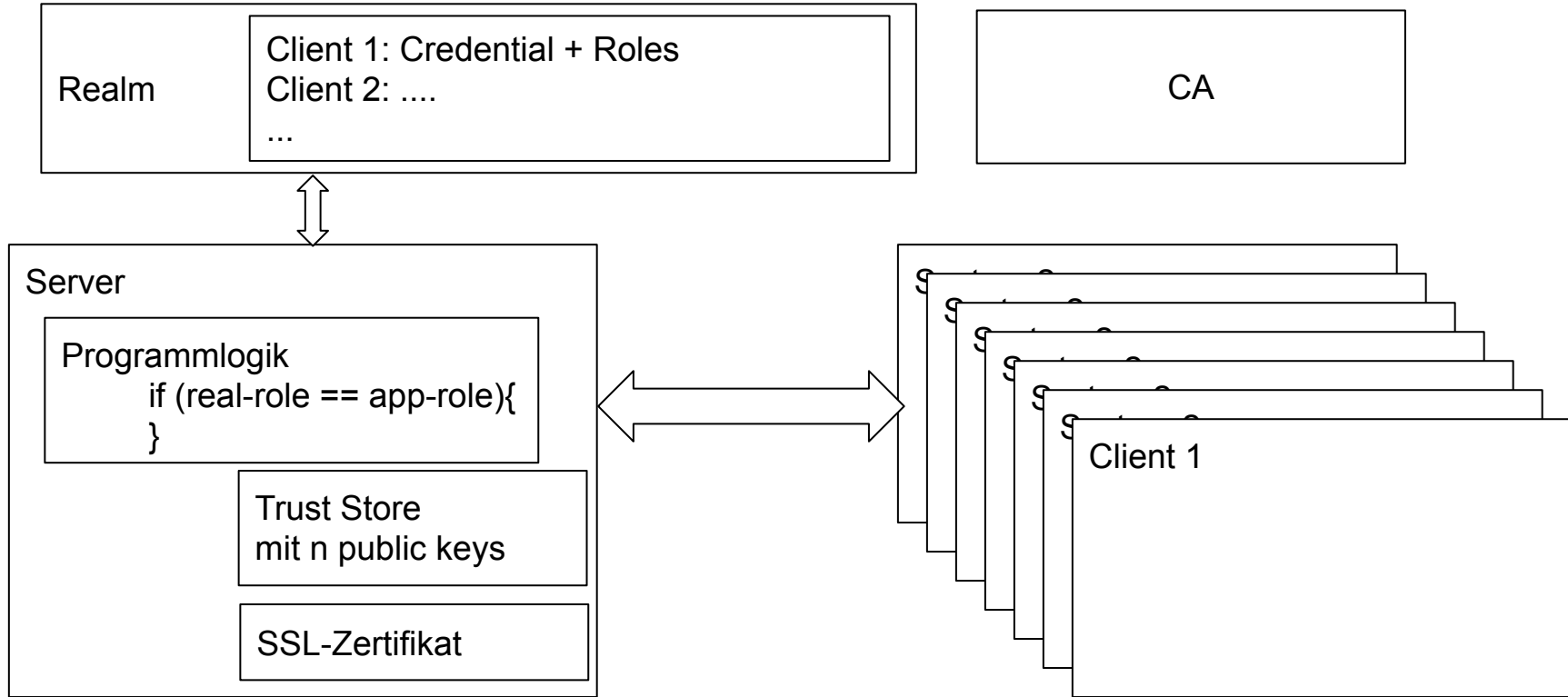
System-Architektur auf Basis von SSL-Zertifikaten



- Pros
 - Funktioniert und ist sicher
- Cons
 - Passt nicht so ganz auf die http-Infrastruktur
 - Beträchtlicher Aufwand zur Erzeugung und Installation von Zertifikaten
 - Zusätzliche Komplikation durch Expiration
 - Ausstellung von Zertifikaten für real existierende Personen führt zu einer Inflation der Anzahl der nötigen Zertifikate
 - Autorisierung ist komplett unbetrachtet

- Wird eingesetzt zum Aufbau einer sicheren Kommunikation
 - Vereinfachung: Nur ein Kommunikationspartner installiert ein Zertifikat
 - Zertifikate werden nur auf den internen Systemen (Web Server, Datenbanken etc.) eingerichtet
- SSL ist damit keine Komplettlösung
 - Authentifizierung von “Clients” sowie die Autorisierung müssen “anders” gemacht werden

- Einführung eines so genannten “Realms”
 - Benutzer-Informationen
 - Benutzer-Name
 - Credential (Zertifikat, Password, ...)
 - Zugewiesene Rollen
- Übertragung der Client-Informationen
 - BASIC
 - Benutzername und Password
 - FORM
 - Benutzername und Password als Formular-Parameter
 - CERTIFICATE
 - Nachweis der Identität



Organisationen

OWASP

- Open Web Application Security Project
- OWASP Top 10
 - Liste der Top-Security-Lücken
 - Aktualisiert alle paar Jahre
 - Kriterien für die Platzierung
 - Häufigkeit
 - Impact

- SQL Injection
 - Der Klassiker: Hugo' or 'true'='true
 - Als Angreifer habe ich ab jetzt die Kontrolle über die Datenbank aus Sicht des Datenbank-Benutzers
 - select, insert, update, (delete)

- SQL Injection funktioniert nur bei Verwendung von Statements
 - PreparedStatement ist erst einmal sicher
 - Don't:
 - `@Autowired private DataSource dataSource;`
- Diskussion Native Query
 - `String sql = "..." + param + "..." //Don't`
 - `Query q = entityManager.createNativeQuery(sql);`
 - `q.setParameter(...) //OK`
- Diskussion mit `createQuery()`
 - `String jpaQl = "..." + param + "..." //Don't`
 - `Query q = entityManager.createQuery(jpaQl);`
 - `q.setParameter(...) //OK`

- Lösung
 - Code Review
- oder
 - Parameter-Validierung
 - Funktioniert nur mit Validierungs-Framework
 - `if (param1.contains("")) throw new IllegalArgumentException()`

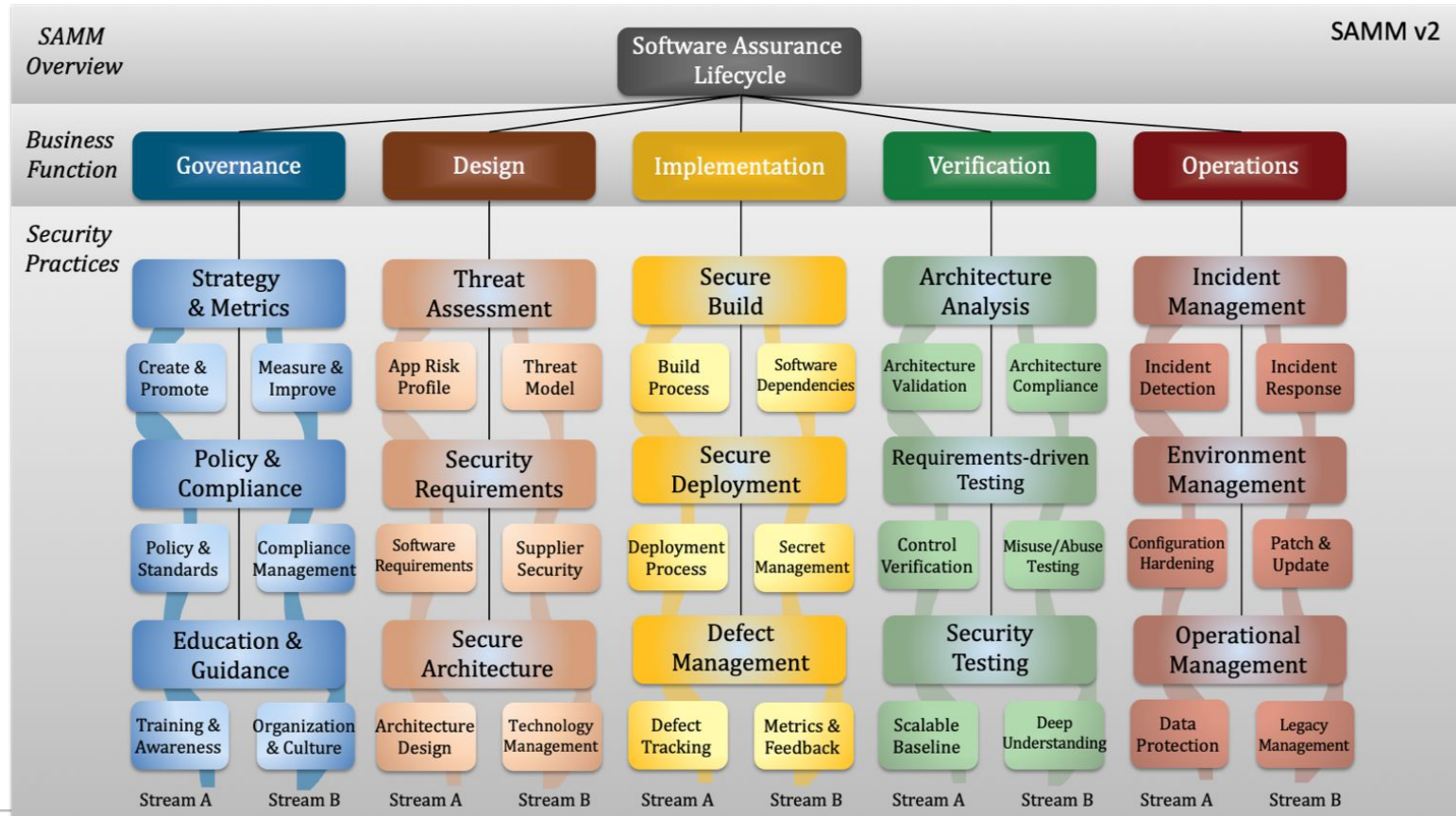
- SQL Injection ist wahrscheinlich gar nicht mehr das Problem...
- Heute:
 - NoSQL
 - XML, JSON
- Wer startet die “Injection”
 - Der klassische Hacker-Client
 - Auch jede Ressource kann zu einer Injection genutzt werden

- Alle Inputs müssen validiert werden
- mit einer White List
- definiert in der zentralen Firewall
 - WAF
- Ressourcen benutzen keine Formate, die Logik enthalten

- Gehen Sie davon aus, dass Passwörter nicht genügend sicher sind
- Im Rahmen von Java/Spring Security größtenteils bereits gelöst
 - Session IDs etc. sind sicher

OWASP ist mehr als die Top 10-Liste!

- Web Goat und Web Wolf
 - Trainings-Umgebung zur Security-Sensibilisierung für Entwickler
 - <https://owasp.org/www-project-webgoat/>
- ESAPI
 - Ein API zur Implementierung einer Web Application Firewall
 - Java-Lösung ist vorhanden
- OWASP SAMM
 - Software Assurance Maturity Model



Weitere Organisationen

- Common Vulnerability Exploits
 - Informationsbasis mit allgemeinen Strategien zum Angriff
 - <https://cve.mitre.org/>
- Att&ck
 - Tool zur Einschätzung: Mit welchen Angriffen ist bei meiner Anwendung wohl zu rechnen
 - <https://attack.mitre.org/>
- Common Weakness Enumeration
 - Sammlung typischer Programm-Fehler
 - <https://cwe.mitre.org/>

<https://nvd.nist.gov/vuln>

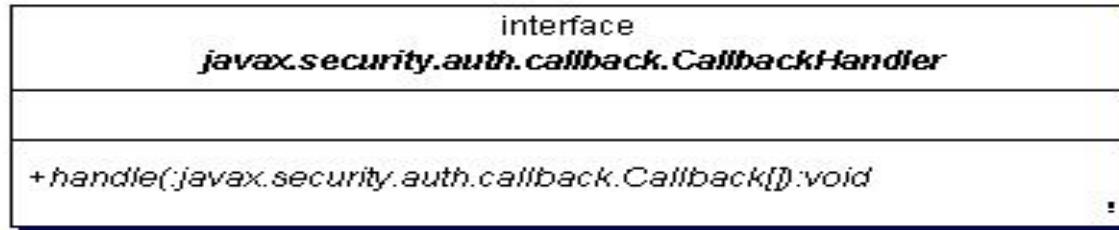
Java

JAAS: Grundlegende Begriffe

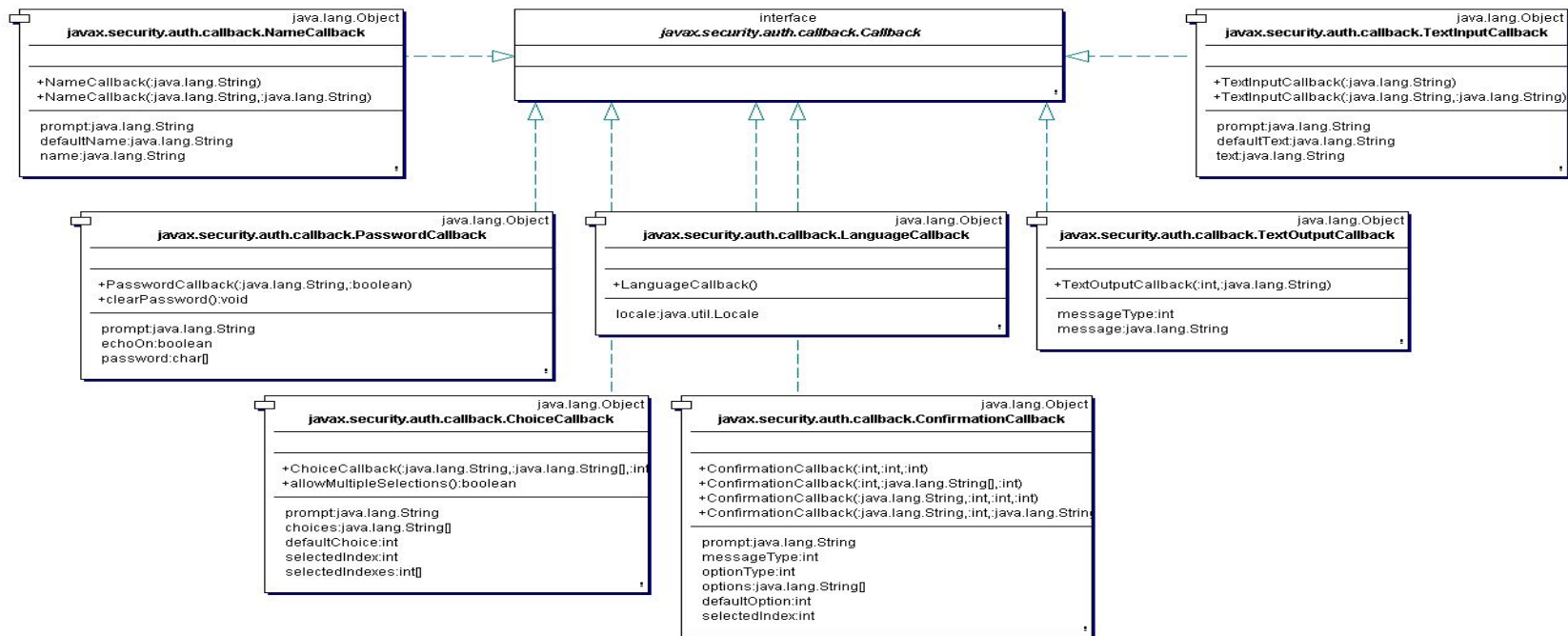
- Das Realm definiert einen eigenen Namensraum für Benutzer und Gruppen. Ein Realm wird persistent verwaltet.
- Ein einzelner Benutzer kann keiner, einer oder mehrerer Gruppen zugeordnet sein.
- Benutzer und Gruppen können z.B. in Tabellen abgelegt werden. Jede Gruppe besteht aus keinem, einem oder mehreren Benutzern. Eine Rolle ist eine Menge aus Gruppen und Benutzern. Rollen sind anwendungsabhängig.

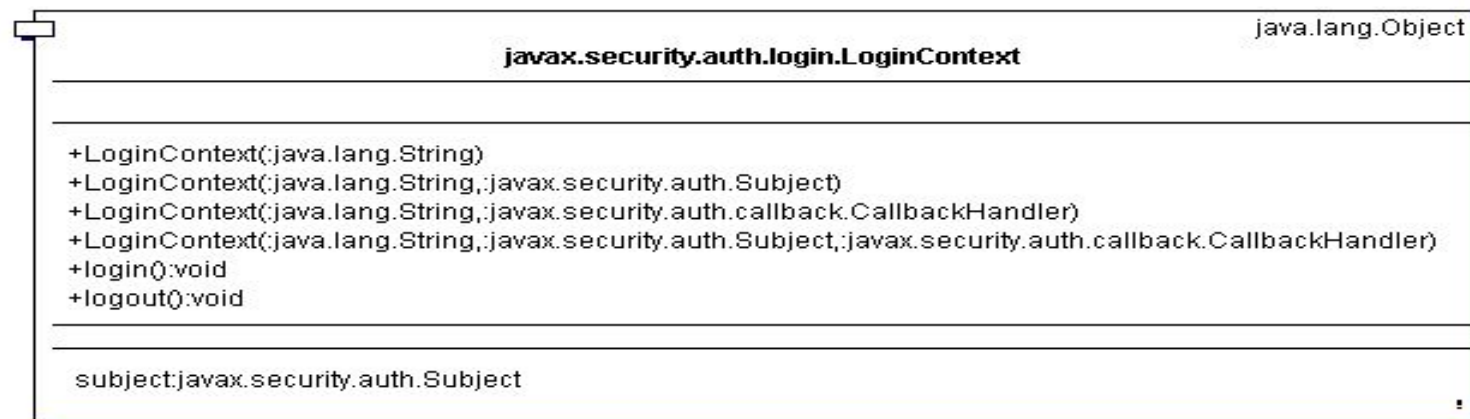
- Prinzipal
 - Ein Name, eine Personalnummer, ein Benutzername
 - Interface `java.security.Principal`
- Credential
 - Ein Credential ist kein Java-Interface sondern irgendeine Klasse, die zur Authorisierung benutzt werden kann
 - Je nach dem verwendeten Authentifizierungsmechanismus kann dies ein einfaches Byte-Array als Password oder ein Zertifikat sein.

- Subject erfüllt drei Aufgaben:
- Halten von einem oder mehreren Principal-Referenzen.
- Halten von Credentials.
- Ablegen eines konkreten Subjects innerhalb des aufrufenden Threads. Dies erfolgt durch den Aufruf einer der statischen Methoden `doAs()` oder `doAsPrivileged()`.
 - Nach diesem Aufruf kann das Subject als Bestandteil des `AccessContext` ausgelesen werden.



- `javax.security.callback.Callback` ist ein reines Marker-Interface
- Callback-Implementierungen dienen zum Sammeln und Transport von Informationen, die zur Authentifizierung benötigt werden.
- Der Authentifizierungs-Mechanismus ruft für den `CallbackHandler` die Methode `handleCallback(Callback)` mit geeigneten Callbacks auf.
- Der `CallbackHandler` prüft den Typ des übergebenen Callbacks und setzt die geforderten Parameter





interface <i>javax.security.auth.spi.LoginModule</i>
<i>+initialize(javax.security.auth.Subject, javax.security.auth.callback.CallbackHandler, java.util.Map, java.util.Map):void</i> <i>+login():boolean</i> <i>+commit():boolean</i> <i>+abort():boolean</i> <i>+logout():boolean</i>

- Der initialize-Methode werden alle benötigten Referenzen übergeben:
 - Das zu ergänzende Subject.
 - Der CallbackHandler des Aufrufenden.
 - Eine Map mit dem „Shared-State“, der allen an der Authentifizierung beteiligten LoginModules gemeinsam ist.
 - Eine Map mit den Konfigurationsparametern.

- Der Authentifizierungs-Teil des JAAS wird durch einen Provider zur Verfügung gestellt:
- LoginModule,
- Principal,
- Credential-Klassen,
- Spezielle Callback-Klassen,
- Callback-Handler

- Für den Client beschränkt sich der JAAS-Provider auf eine Hilfsklasse, die eine Konfigurations-Datei ausliest
 - `javax.security.auth.login.Configuration`
- Eintrag in der Konfigurationsdatei der Java-Laufzeitumgebung für den Default-Provider Sun (
 - `<JAVA_HOME>\jre\lib\security\java.security`
 - `login.configuration.provider=com.sun.security.auth.login.ConfigFile`

- Suchen einer Konfigurationsdatei an Hand eines Eintrags in der JRE-Konfigurationsdatei:
 - `login.config.url.1=file:${user.home}/.java.login.config`

- Auslesen einer Umgebungsvariable
 - `java.security.auth.login.config`

```
SimpleLogin{  
  
};  
SSLLogin{  
  
};  
default{  
  
};
```

```
SimpleLogin{  
    com.hotspots.javax.security.login.spi.SimpleLoginModule;  
};  
SSLLogin{  
    com.hotspots.javax.security.login.spi.SSLLoginModule;  
    com.hotspots.javax.security.login.spi.CertificateLoginModule;  
};  
default{  
    com.hotspots.javax.security.login.spi.SimpleLoginModule;  
};
```

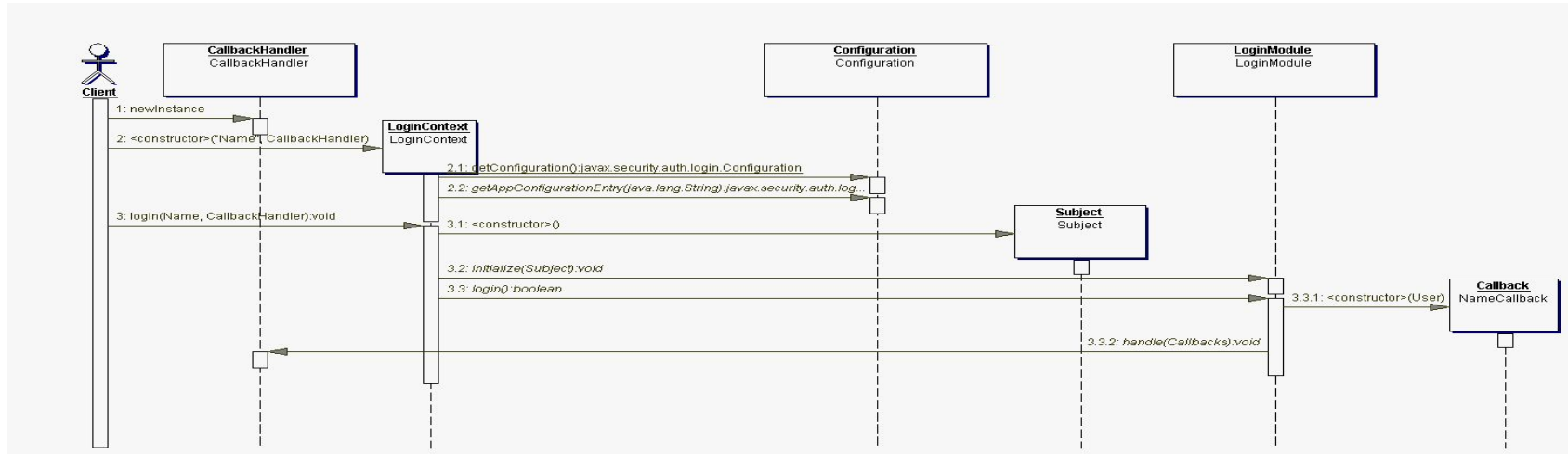
```
SimpleLogin{
    com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
    com.hotspots.javax.security.login.spi.SSLLoginModule debug=true;
    com.hotspots.javax.security.login.spi.CertificateLoginModule debug=true
                                                                    url=ldap://hotspots.com;
};
default{
    com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

Diese Parameter werden als Map der initialize-Methode des LoginModule übergeben.


```
SimpleLogin{
    com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
    com.hotspots.javax.security.login.spi.SSLLoginModule requisite debug=true;
    com.hotspots.javax.security.login.spi.CertificateLoginModule required debug=true
                                                url=ldap://hotspots.com;
};
default{
    com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

required	Erfolgreicher Login notwendig, die weiteren Module werden in jedem Fall abgearbeitet.
requisite	Falls der Login nicht erfolgreich ist, wird sofort abgebrochen.
sufficient	Falls der Login erfolgreich ist, werden keine weiteren Module mehr abgearbeitet.
optional	Das LoginModule kann erfolgreich sein oder auch nicht.

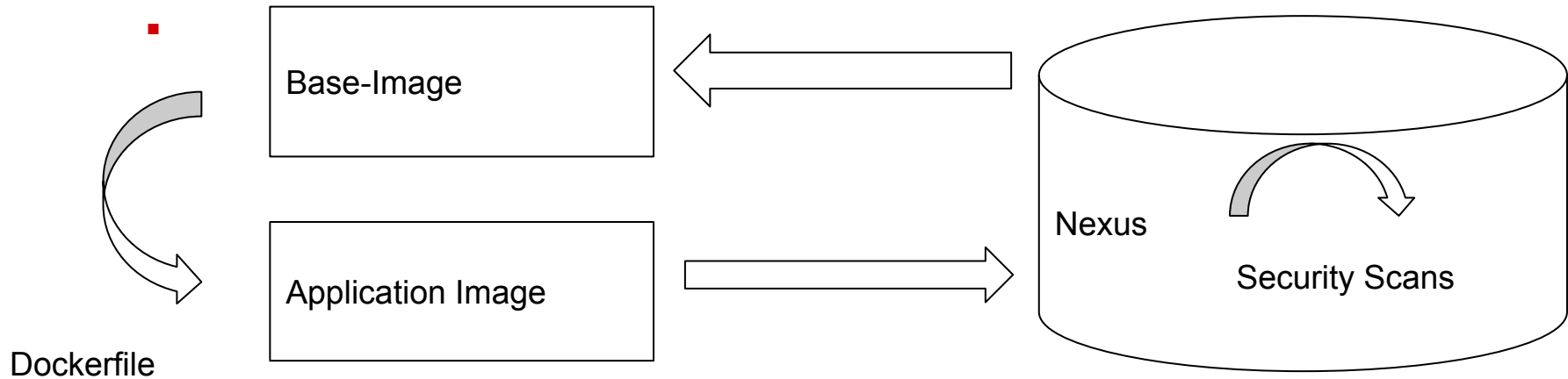
Der LoginContext ist eine Fassade zur Erzeugung eines Subjects



Die Rolle des Build-Prozesses

- Build-Prozess Maven-basiert
 - Dependency Management
 - CI/CD-Prozess berücksichtigt Security Alerts
 - Realisiert über das Artefakt-Repository
 - Nexus “quaranteed” Security-auffällige Bibliotheken
 - Regelmäßige Security Scans
 - Auch die macht das Artefakt-Repository

- DevOps oder “OpDevs”
- State of the Art: Docker
- Artefakt der Laufzeitumgebung wird ebenfalls über einen Build-Prozess definiert



- Injection-Tests
- Penetration-Tests
- ...

- Smoke-Tests
 - Getestet wird Logging und Monitoring
- Security Audits

Spring Boot

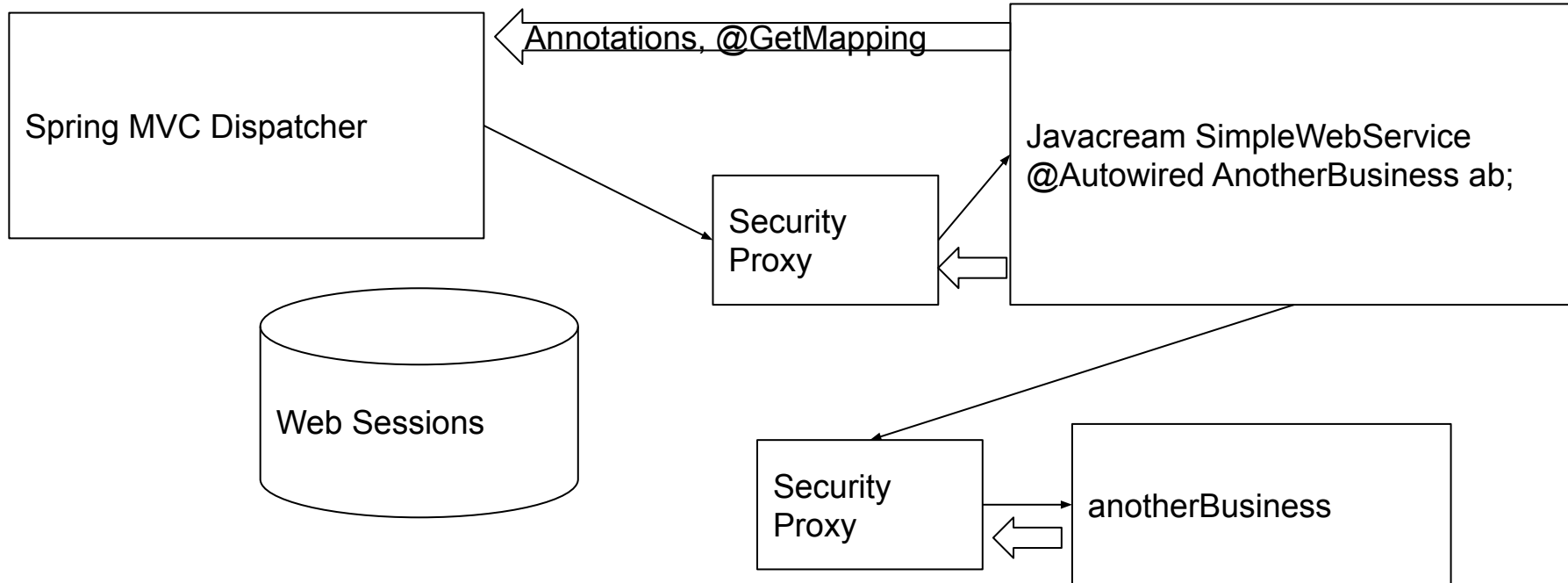
- Definiert einen getesteten Stand von Dependencies
 - Spring Boot Starter
- Integration des owasp dependency checks
 - Technik:
 - <https://nvd.nist.gov/vuln> hält eine aktuelle Liste von Vulnerabilities
 - Maven-Plugin Download der aktuellen Liste (einige 100 MB)
 - Abgleich der Dependencies im POM gegen diese Liste

- Prozessdefinition einer Spring Boot Application
 - `java -jar myapp.jar`
- ToDo: Umsetzung von Least Privileges?
 - Kann geregelt werden vom Environment (Betriebssystem, Firewalls)
 - Spezieller App-User wird eingerichtet und diesem werden die nötigen Berechtigungen gegeben
 - Operations
 - Automatisieren z.B. mit Ansible Playbooks
 - In Java ist der konfigurierbare SecurityManager vorhanden
 - `java.policy` enthält die Berechtigungen
 - Development
 - Virtualisierung oder Container
 - z.B. VMWare oder Docker
 - DevOps

Spring Security

- Fokus auf Web Anwendungen
 - RESTful WebServices
 - Klassische Web Anwendungen
 - Klassische Servlet Filter
 - Prüfe: Authentifizierung Nein/ja
 - Redirect auf eine Login-Seite
 - Weiterdelegieren an die aufgerufene Web Seite/Service

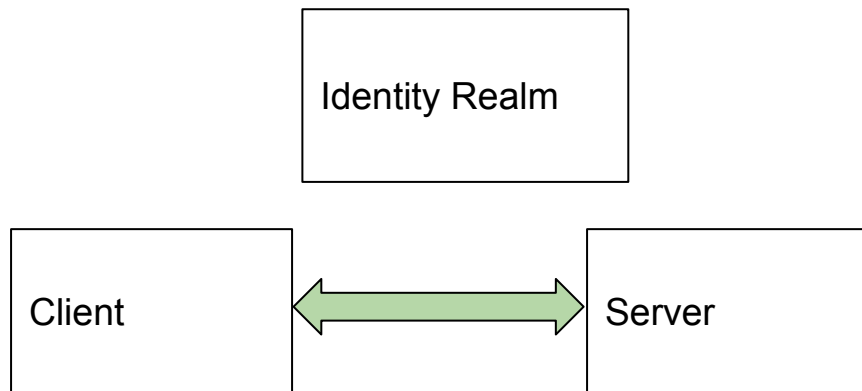
- Fokus auf Fachobjekte

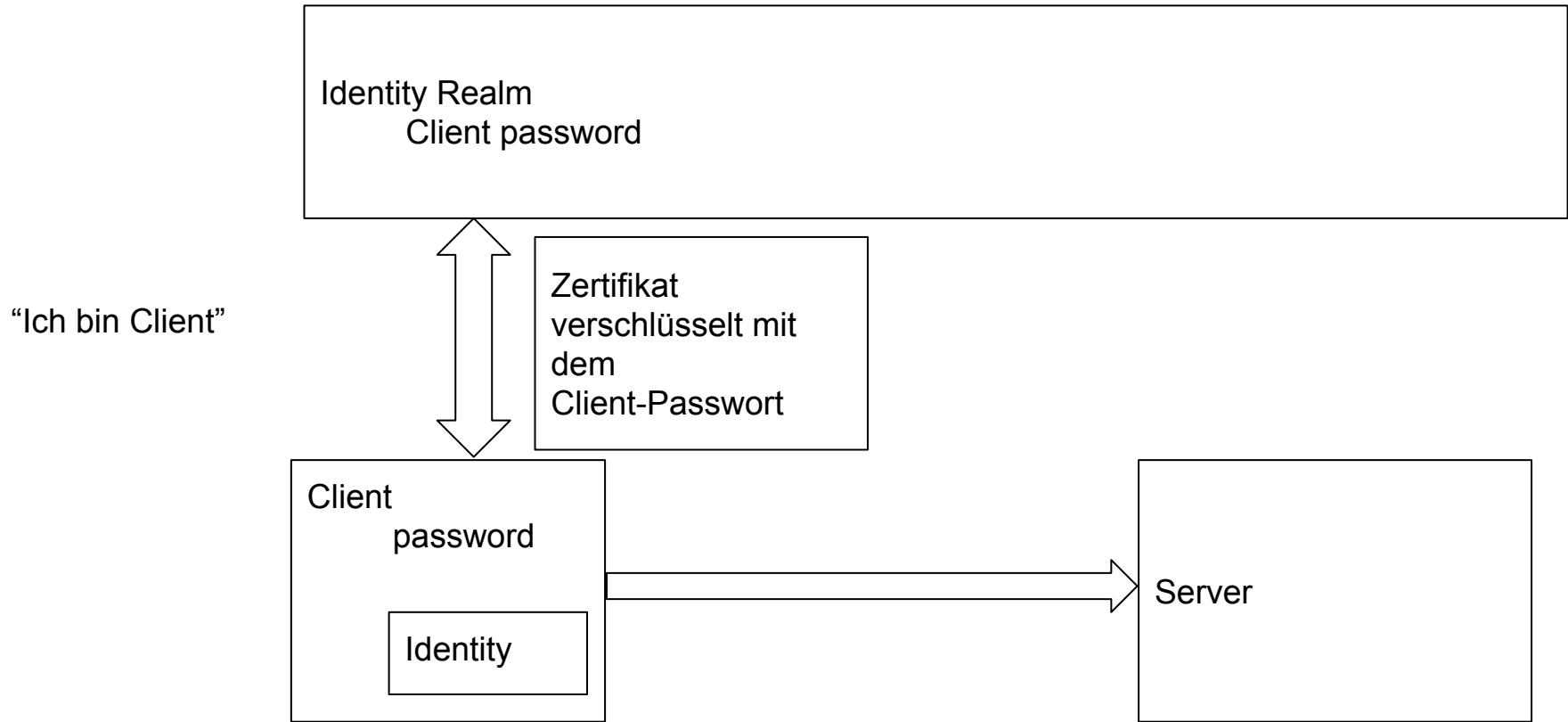


Identity Management

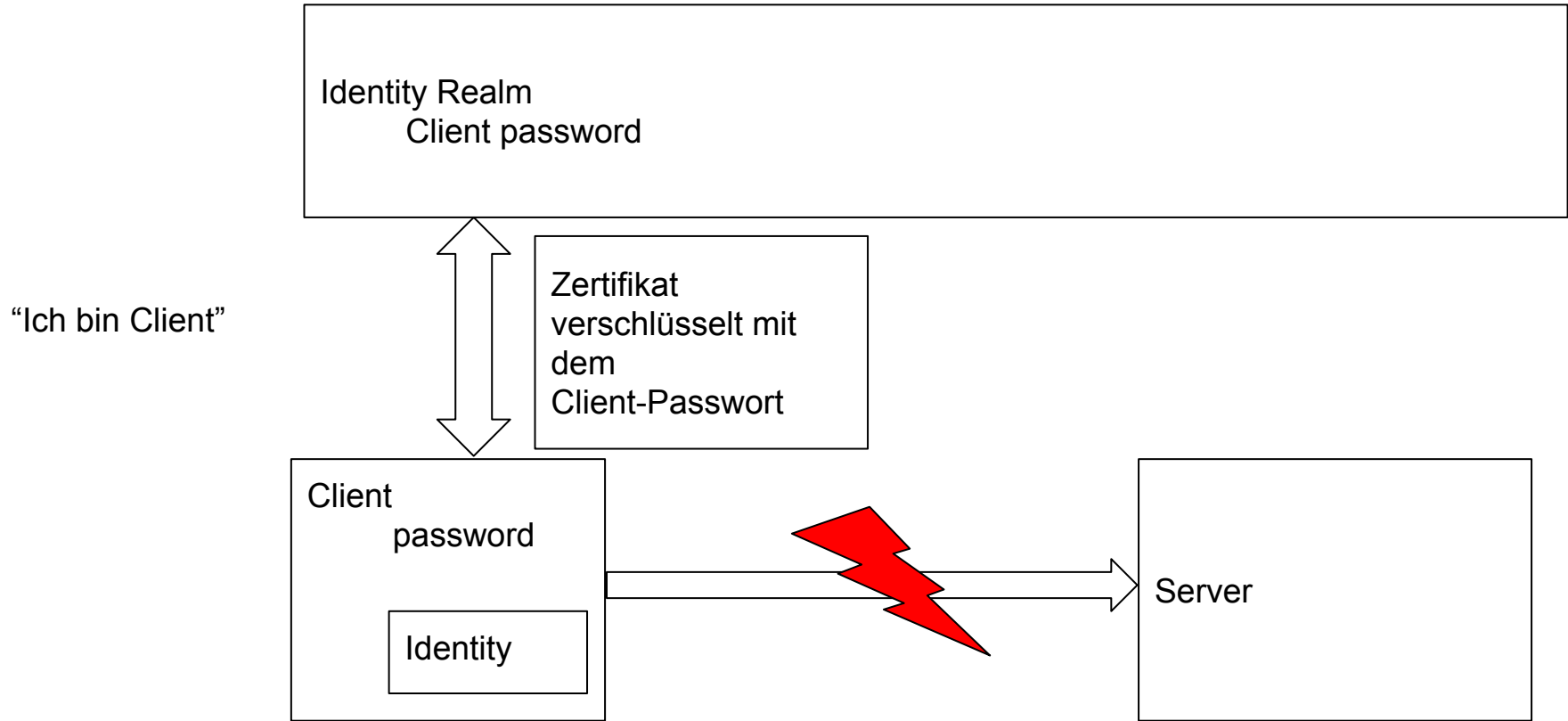
Kerberos

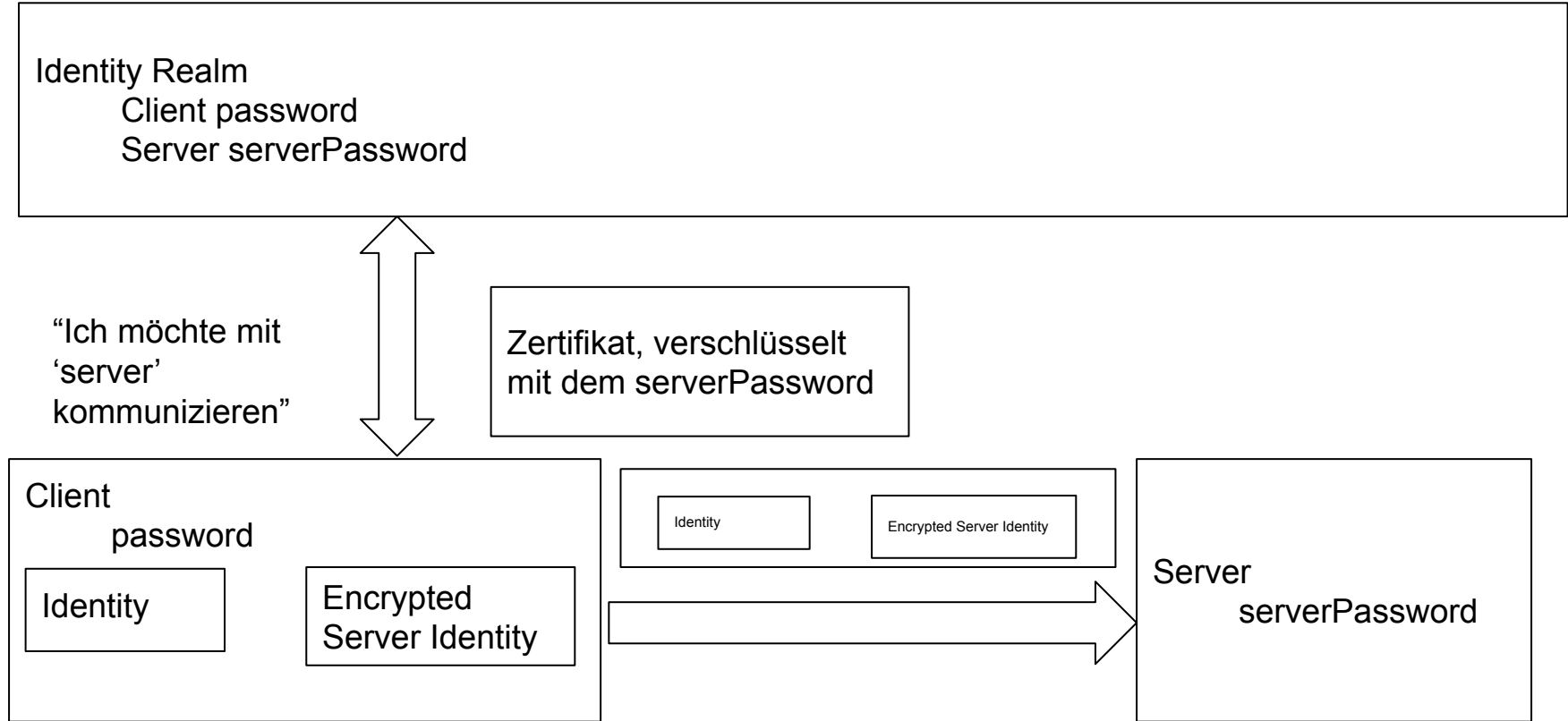
- Kommunikation erfolgt über eine unsichere Verbindung
 - Es werden niemals Credentials über das Kommunikationsprotokoll übertragen
- Alle Daten werden immer in verschlüsselter Form übertragen
- Identities werden in einem Realm verwaltet



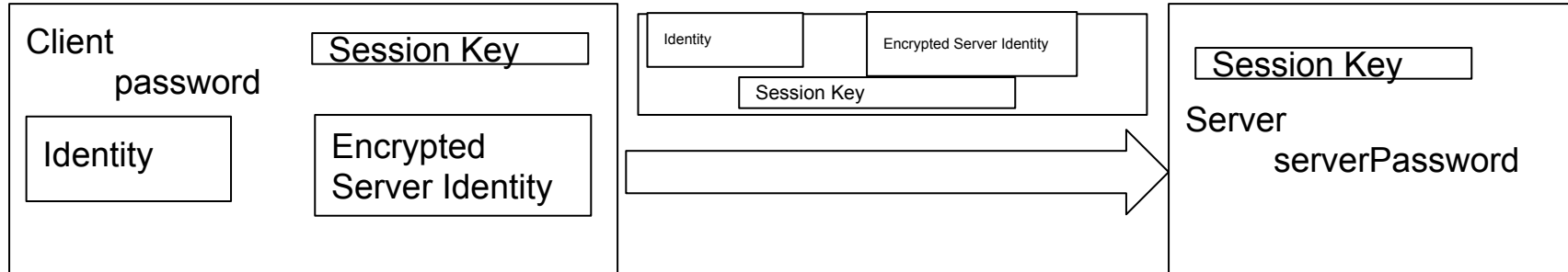


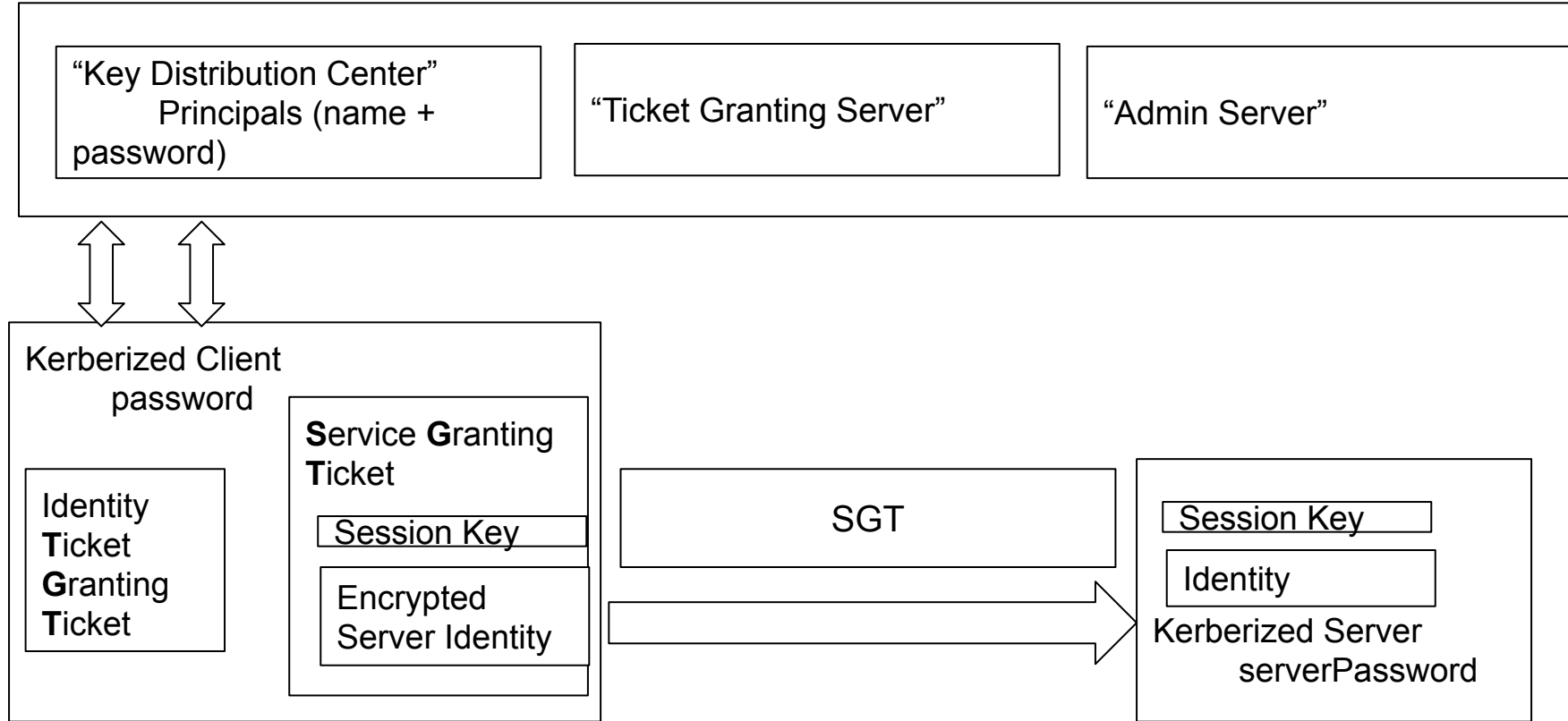
Umsetzung: Naiv: Damit wäre ich fertig





Identity Realm
Client password
Server serverPassword





- Mit Kerberos ist ein zusätzlicher Einsatz von SSL nicht notwendig
- Autorisierung in Kerberos ist nicht vorgesehen
 - Kerberos alleine ist kein Komplettpaket
 - In der Praxis:
 - Kombination mit einem Rollen-Realm
 - Häufig ein Directory-Server, LDAP
- Kerberos ist eine “Consortium-Spezifikation”
 - MIT
 - Implementierungen von MIT, Heimdal, Windows Domain Controller
 - Diverse schlanke Implementierungen, z.B. auch für Test-Umgebungen
- “Kerberizing” einer Anwendung kann sich auf das Generic Security Services Application Programming Interface” stützen, (GSSAPI)
 - Verfügbar für alle gängigen Programmiersprache

Kerberos: Umgebung

- insbesondere ein KDC
 - Windows Domain Controller
 - Standalone-Installation, z.B. MIT
 - Docker Images mit KDC-Installation sind im Docker Hub verfügbar
 - Test-Umgebung im Spring-Umfeld: “MiniKdc”
- Zur Präsentation
 - `docker run -d --name krb5_2 -e KRB5_REALM=JAVACREAM.ORG -e KRB5_KDC=localhost -e KRB5_PASS=xxxxxx -p 88:88/tcp -p 88:88/udp -p 464:464 -p 749:749 -v /root/kerberos/keytabs:/keytabs gcavalcante8808/krb5-server`

- Realm-Namen in Kerberos immer ausschließlich in Großbuchstaben
 - JAVACREAM.ORG
- Kerberos hat ein Rollen-Konzept mit so genannten Access Control Lists
 - ausschließlich benutzt für die interne Administration
 - der Kerberos-Administrator legt Principals an
- Kerberos-Server bieten lauschende Ports an
 - 88 (KDC)
 - TCP oder UDP
 - 464 (Admin)
 - 749 (Ticket Granting Server)
- Meine Präsentation: `h2908727.stratoserver.net:88`

- <name>@DOMAIN
- <name>
 - Hierarchisch aufzubauen
 - Unterscheide “Benutzer” und Services
 - Beispiel für Benutzer
 - sawitzki@JAVACREAM.ORG
 - sawitzki/admin@JAVACREAM.ORG
 - Beispiel für Services
 - HTTP/host@JAVACREAM.ORG

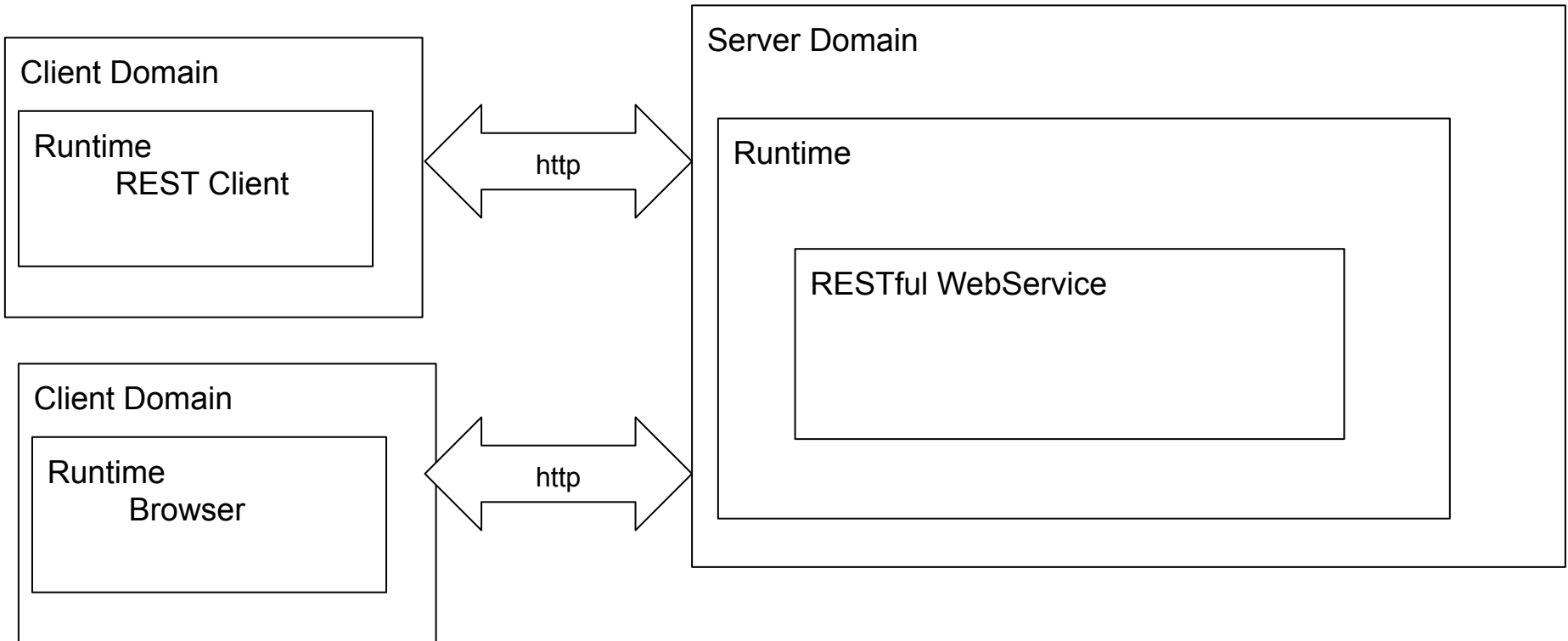
- Kerberos User Client
 - enthält ein paar Utilities
 - klist
 - kinit sawitzki
 - kdestroy
 - Konfiguration /etc/krb5.conf
- Laufzeitumgebung mit vorhandener GSSAPI-Unterstützung
 - z.B. Java-Umgebung enthält GSSAPI in den Standard-Bibliotheken

- JAAS
 - Java Authentication and Authorization Service
- Zur Ausführung wird ein konkretes LoginModule benutzt
 - hier `com.sun.security.auth.module.Krb5LoginModule`

- Diskussionswürdig
 - “Das TGT ist eine Security-Lücke”
 - Mehrwert: Single Sign On
- Alternativ kann eine reine In-Memory-Lösung genutzt werden
- Zur Minimierung eines potenziellen Schadens
 - Expiration (typischerweise ein Arbeitstag)
 - Eine Änderung im KDC (“Sperren eines Benutzers”) führt dazu, dass keine neues Service Tickets angefordert werden können
 - Auch die Service Tickets haben eine Expiration (eher halbe Stunde)

- Starten und Stoppen des Services ist orchestriert
 - Eine Initialisierung zur Erzeugung eines TGT ist im Vergleich zu einem Client so sehr sperrig
- Authentifizierungsinformationen werden als Datei (!) außerhalb des Ticket-Caches gehalten
 - Keytab

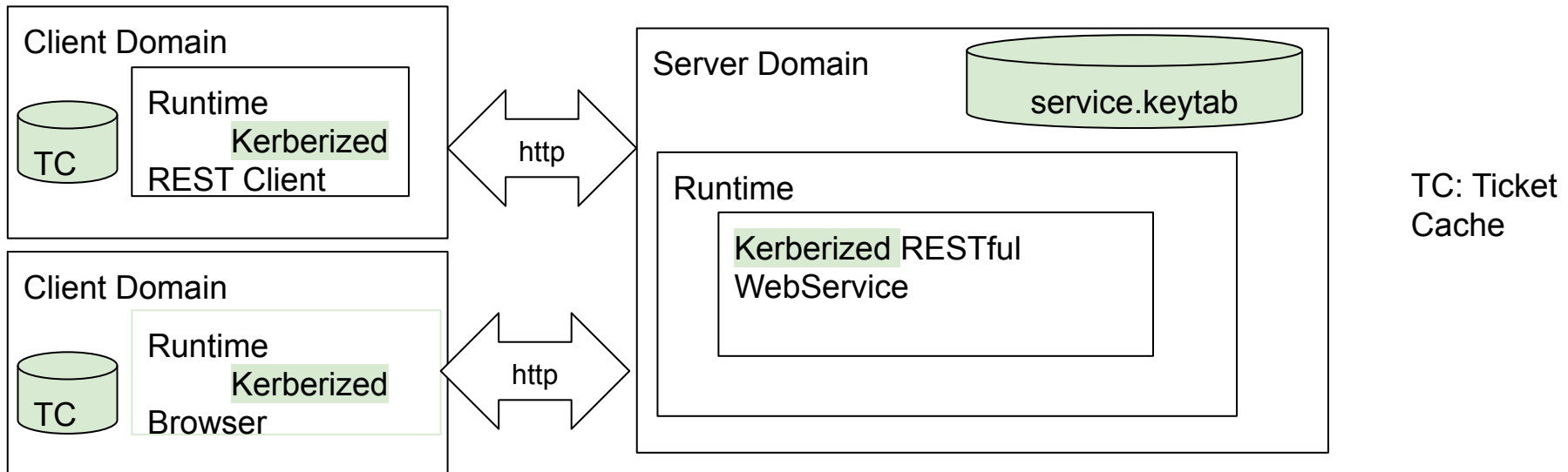
Eine komplett ungesicherte Anwendung



Eine Anwendung mit Kerberos-Authentifizierung

KDC

Principals für Client Domain und deren User sowie den HTTP-Service der Server-Domain



- Simple and Protected GSSAPI Negotiation

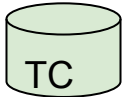
Eine Anwendung mit Kerberos-Authentifizierung im Beispiel

KDC

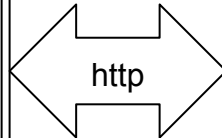
Principals für Client Domain und deren User sowie den HTTP-Service der Server-Domain

kinit javacream/test

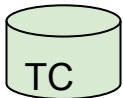
Client Domain



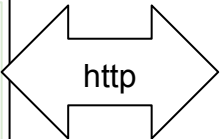
Runtime
Kerberized
REST Client
curl --negotiate -u :



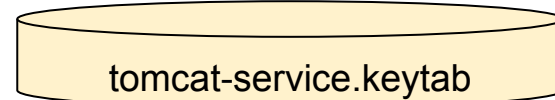
Client Domain



Runtime
Kerberized
Firefox
(about:config)



Server Domain

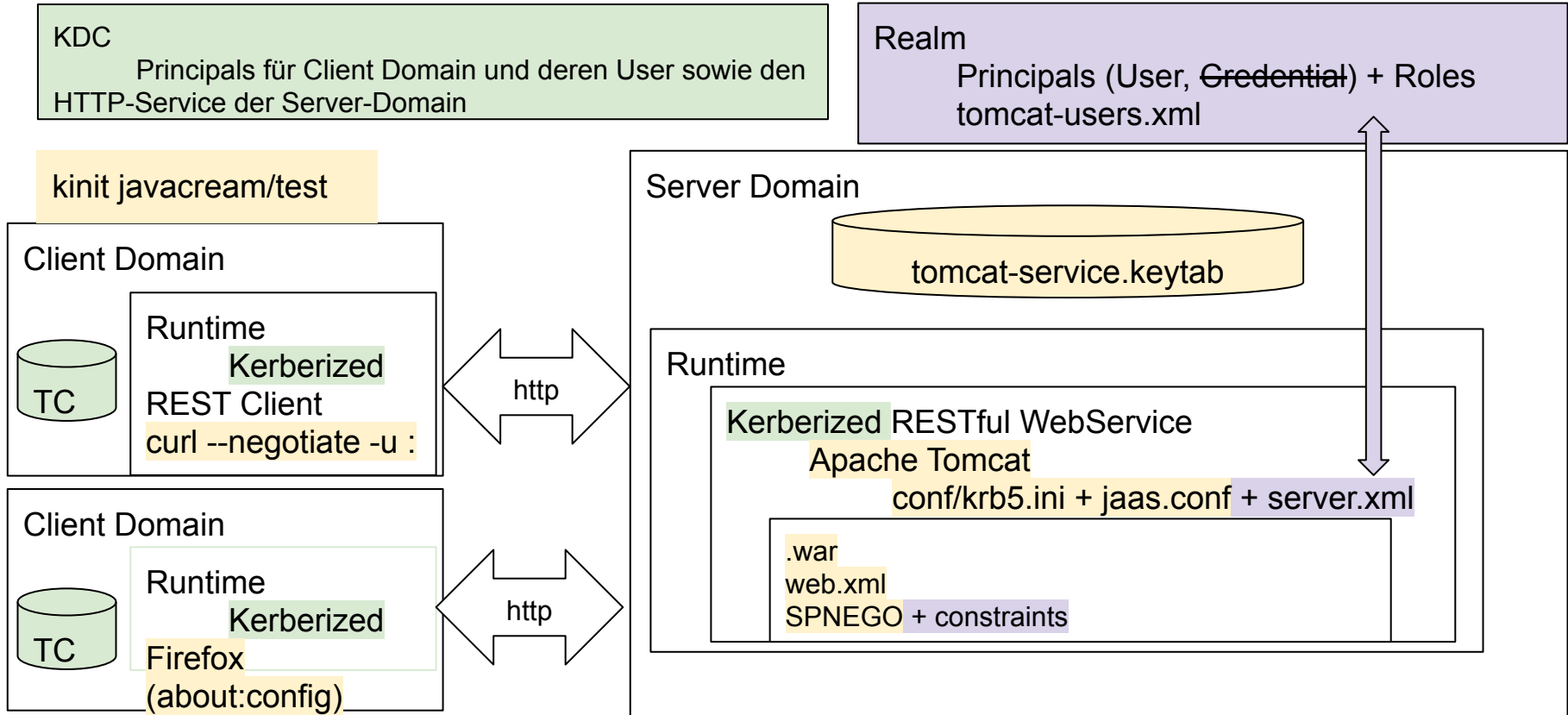


Runtime

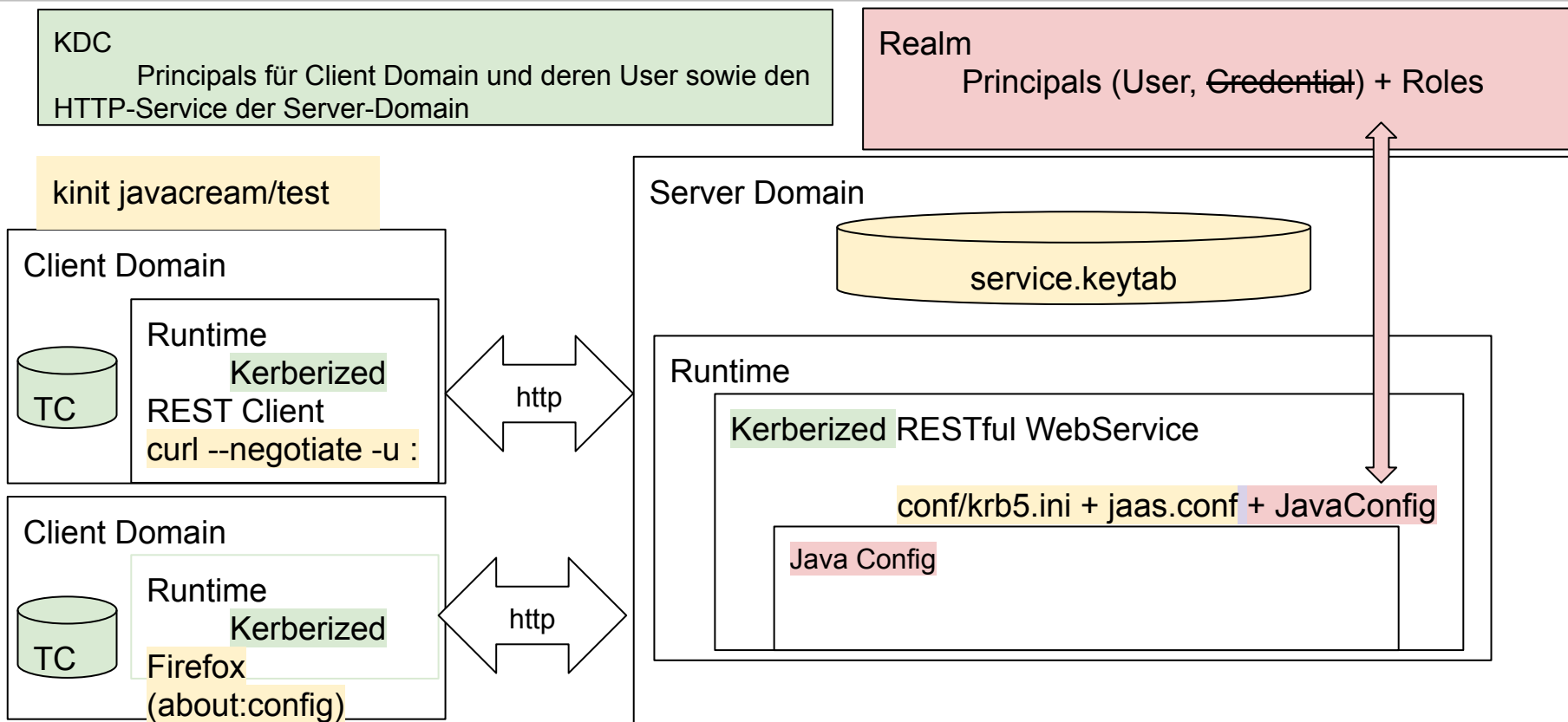
Kerberized RESTful Webservice
Apache Tomcat
conf/krb5.ini + jaas.conf

.war
web.xml
SPNEGO

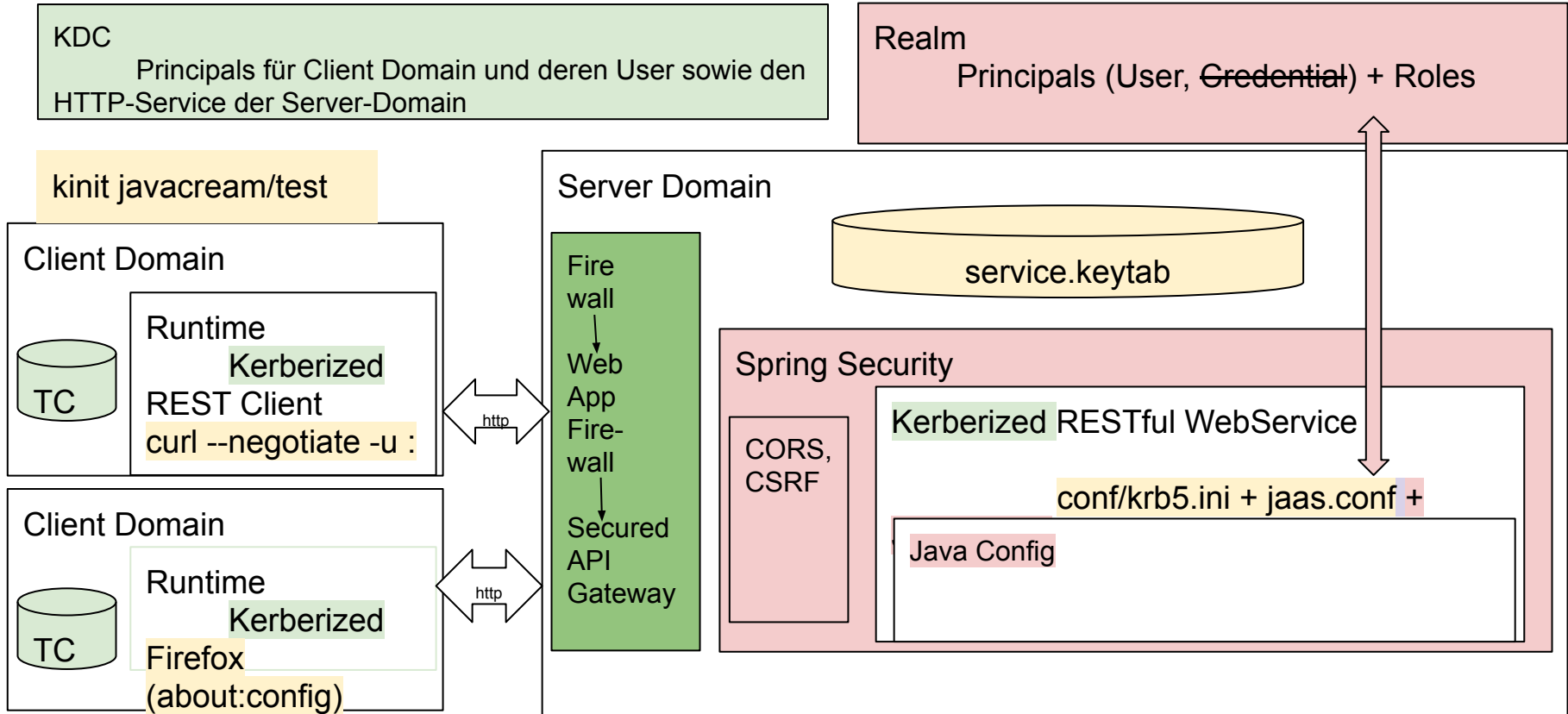
Eine Anwendung mit Kerberos-Authentifizierung im Beispiel + Authorization mit Standard JEE



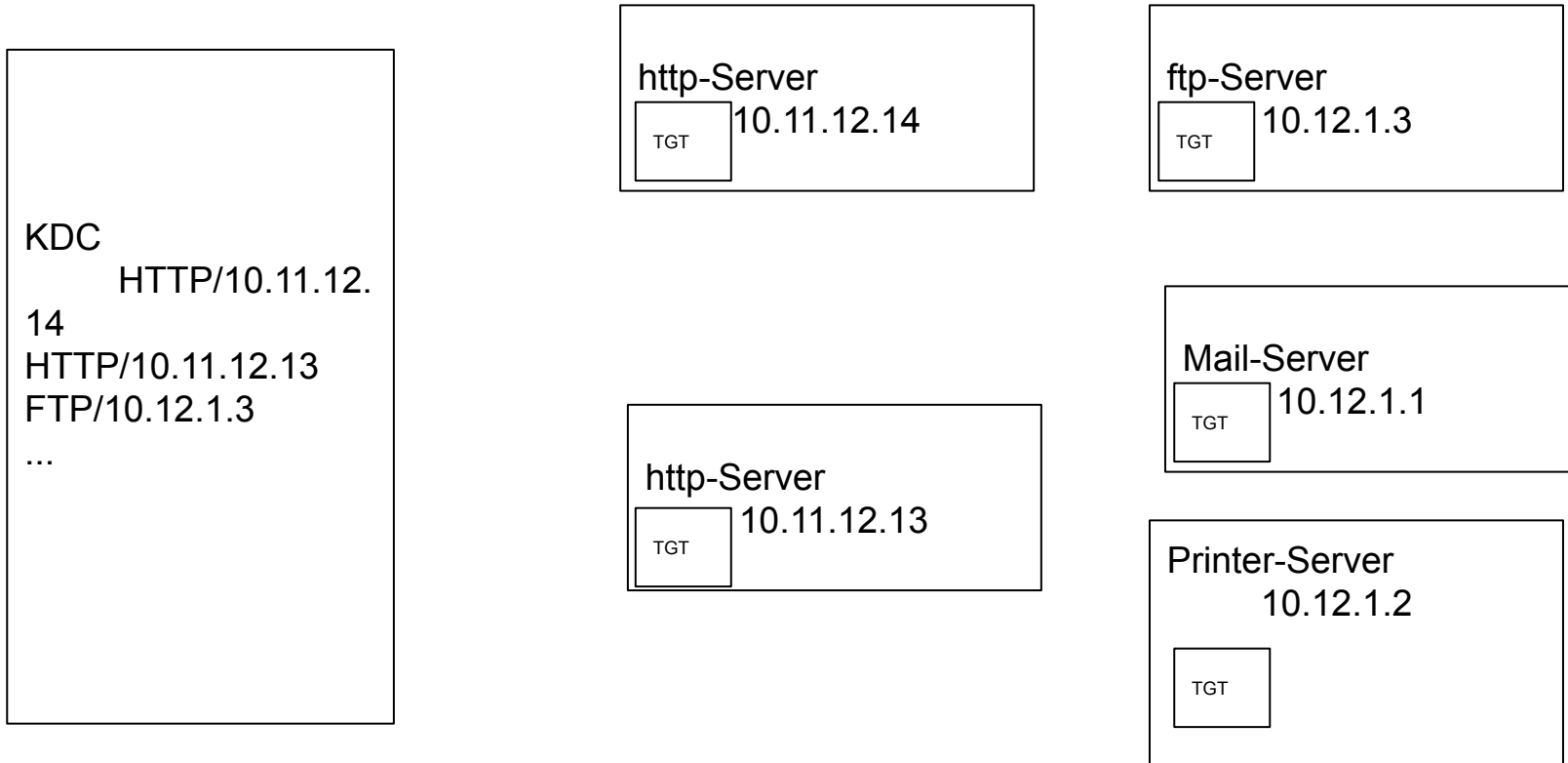
Eine Anwendung mit Kerberos-Authentifizierung im Beispiel mit Spring / Boot / Security



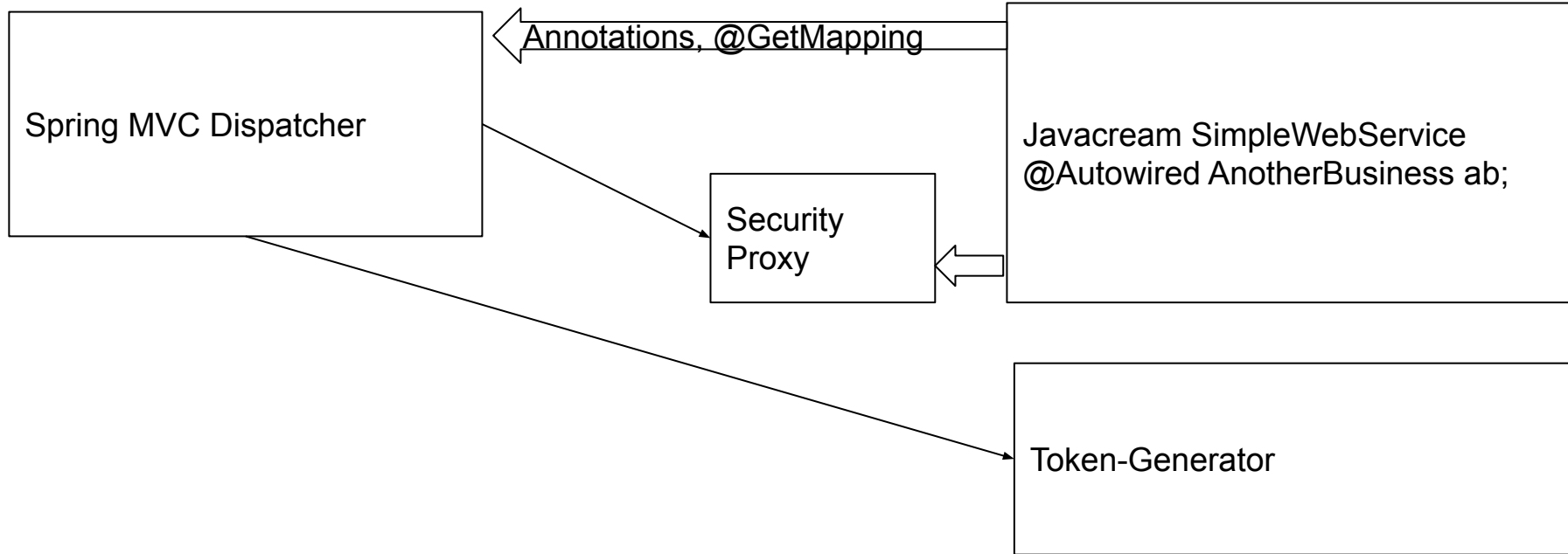
Eine Anwendung mit Kerberos-Authentifizierung im Beispiel mit Spring / Boot / Security+Infrastructure



- Expiration
 - Es gibt auch “renewable” Tickets
- Forwardable
- Historie der Supported Crypto-Algorithmen
- Encryption Types
 - möglichst modern (kein ‘weak’ oder ‘deprecated’)
 - Kerberos-Umgebung auf Client/Host muss diese Algorithmen unterstützen
 - VORSICHT: Browser-Versionen
- Das Ticket Granting Ticket ist die Identität
 - Erlangen des TGT ist der Anmeldevorgang, Lebensdauer definiert die Anzahl der notwendigen Authentifizierungen
 - Mit Kerberos kann SSO implementiert werden

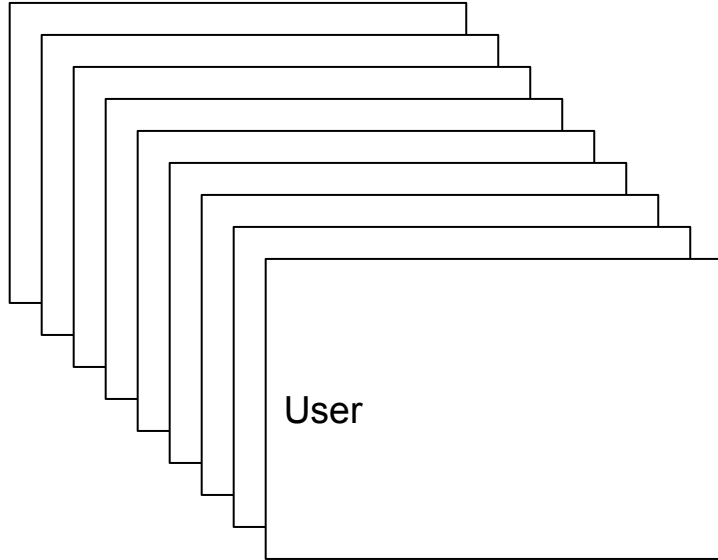


Token-basierte Authentifizierung und Autorisierung

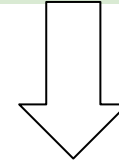


- Token ist ein JSON-Dokument
 - Wohl-definierter Aufbau
 - sub:
 - exp
 - Zugeordnete Rolle(n)
 - Signatur
 - Encodierung z.B. byte64

Einführung in OAuth2



- Profile-Informationen?
- Self-Registration?
- Rollen sollen nicht vom Service aus einem externen Realm gezogen werden müssen
- Öffentlich zugreifbare Authentifizierungs- und Autorisierungs-Instanz



OAuth2 und OpenIdConnect

OAuth2 ist der Ticket-Server

Tickets enthalten auch Role-Informationen

Format: JSON als Json Web Ticket JWT

OpenIdConnect

Der optionale öffentliche Authentifizierungs-Server

Ablage der Profil-Informationen

- RESTful API mit Zugriff über https
- Authorization Endpoint
 - “Normale” Anmeldung des Users/Resource Owners unter Verwendung seiner Credentials
 - Erfolgreicher Authentifizierung liefert als Ergebnis einen so genannten “Authorization Code”
 - Wird der Client-App zur Verfügung gestellt
- Token Endpoint
 - Erzeugt ein “Access Token” als Reference Token
 - Client-Applikation muss sich mit Client-Credentials authentifizieren
 - ClientID + “Client Secret”
 - Mit Hilfe des Authorization Codes wird das Access Token generiert

Vorsicht: Wir müssen Voraussetzungen schaffen!

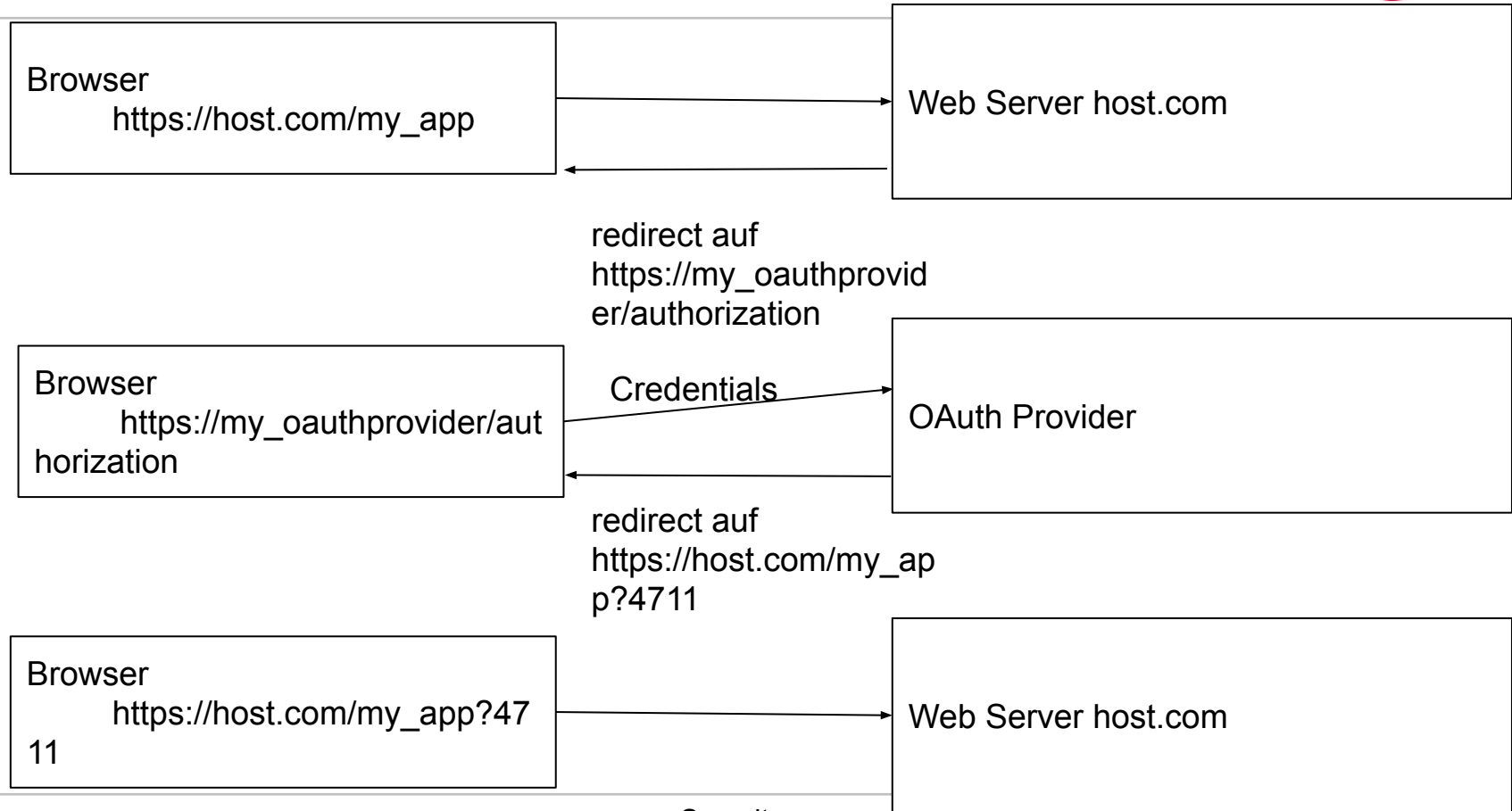
- Als OAuth Provider Admin muss
 - Der User eingerichtet werden
- Administrative Einrichten eines “Clients”
 - ClientID
 - ClientSecret
 - Hinzufügen der User
 - Definieren der Rollen

- Token Validierung
 - Access Token + Angeforderten Rollen
- Self Registration
 - Automatisches Erstellen des Clients ohne administrative Aktionen

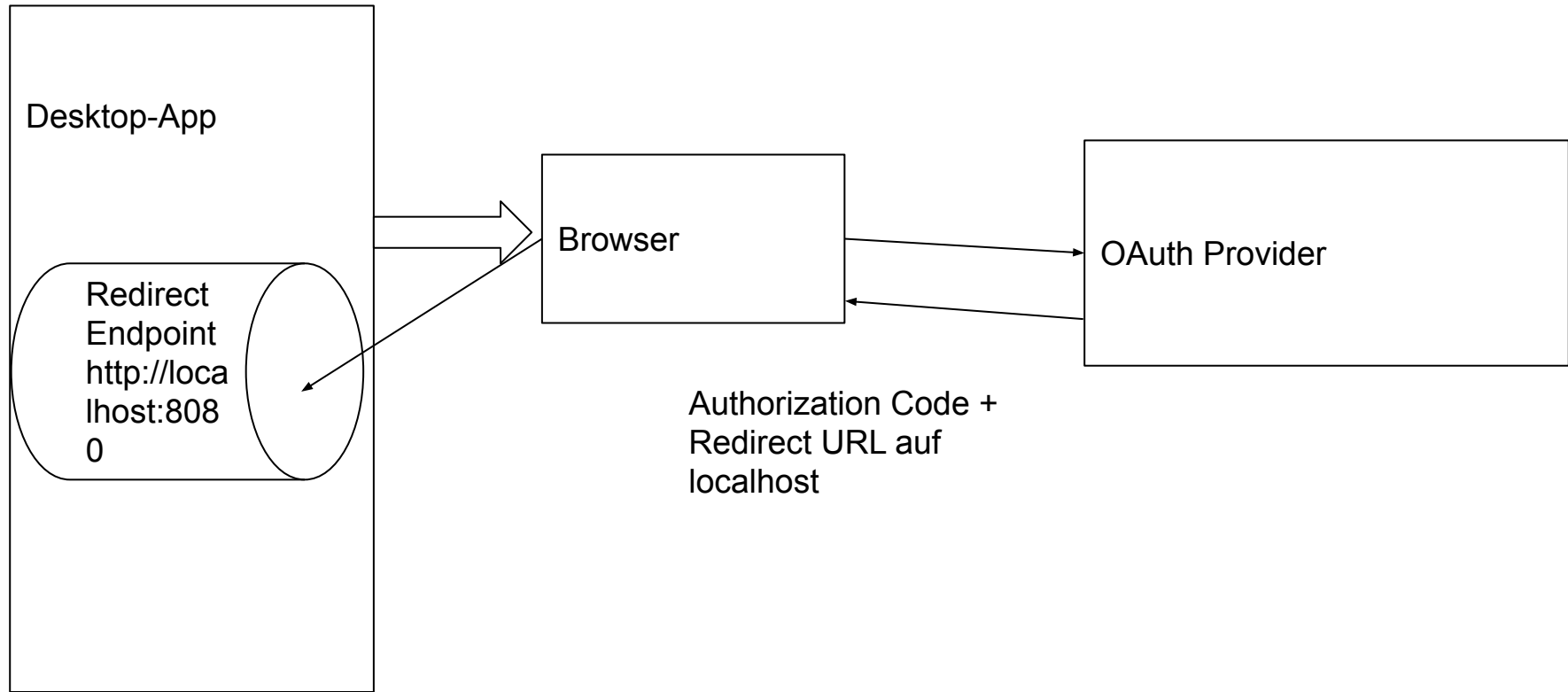
- Redirect Endpoint
- Bestandteil des Clients (!)
- Aufgabe:
 - Anmeldung des Resource Owners soll außerhalb der Client-App erfolgen
 - Client-App bekommt niemals Zugriff auf die Credentials des Resource Owners!

- https://host.com/my_app
 - Resource Owner/User ruft diese Anwendung über eine URL auf
- my_app beendet sich und delegiert weiter an den Authorization Endpoint des Providers
 - https://my_oauthprovider/authorization?redirect_uri=https://host.com/my_app
 - Resource Owner meldet sich an und bekommt den Authorization Code, z.B. 4711
 - Redirect auf https://host.com/my_app?code=4711
- “Restart” von my_app
 - https://host.com/my_app?code=4711
 - Intern wird my_app sich nun an den Token Endpoint wenden und eine Access Token

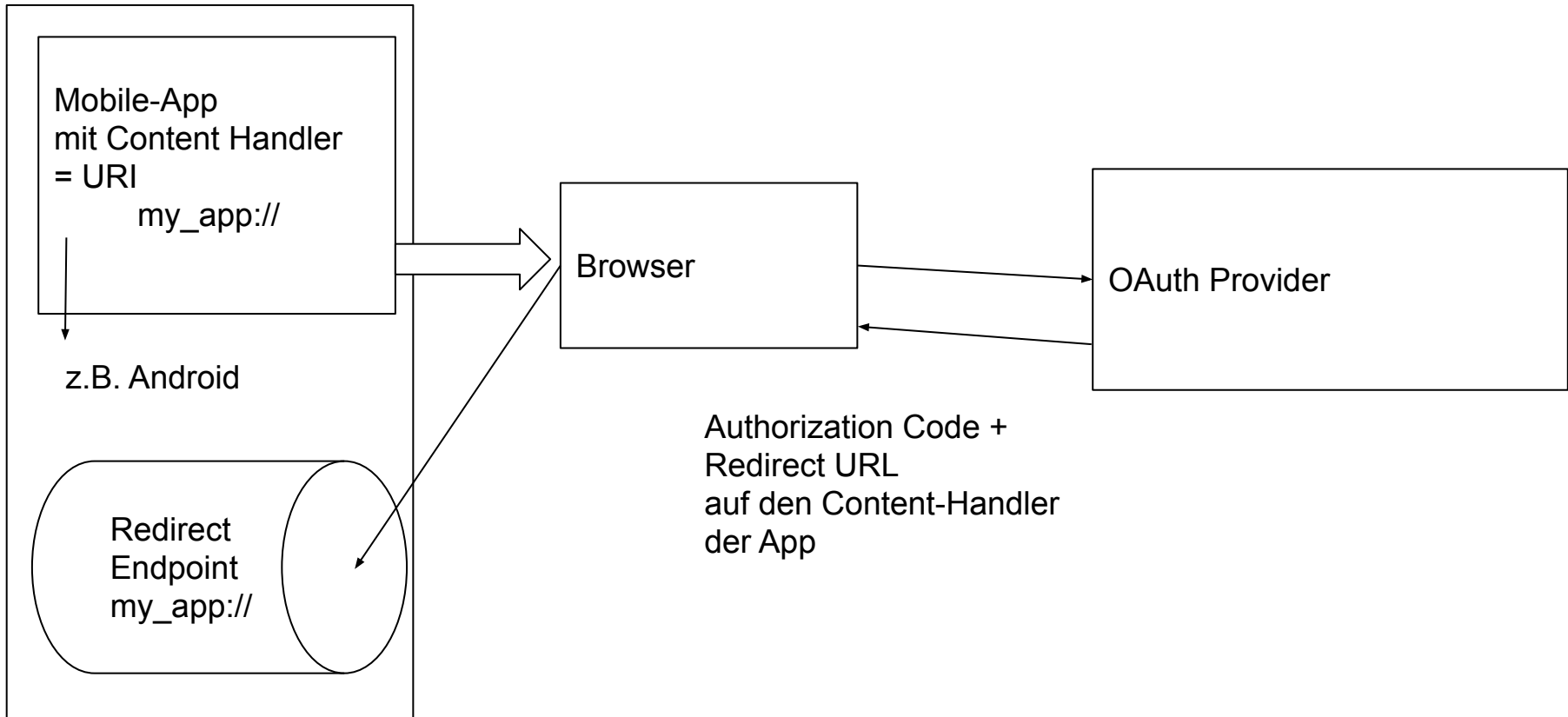
Redirect Endpoint im Browser



Redirect Endpoint in einer Desktop-Applikation



Redirect Endpoint Mobile App



OAuth2 Flows

- Voraussetzungen
 - Client ist im Provider angelegt (OAuth Provider Admin)
 - Ressource Owner ist dem Client zugeordnet
 - Für den Client sind Anwender-Rollen definiert
- Anforderungen an den Client
 - ClientID und Client Secret müssen bekannt sein
 - Sichere und dauerhafte Ablage muss garantiert sein
- Ebenen der Verifikation
 - Provider-Identität ist über das SSL-Zertifikat des Servers prüfbar
 - Resource Owner wird über seine Credentials verifiziert
 - Clients über Client ID und Client Secret
- Client-App hat niemals Zugriff auf die Credentials des Resource Owners

- Authorization Code
 - Wenige Sekunden
- Access Token
 - Wenige Minuten bis hin zu Stunden
 - Analog zur Session ID
- Optional: Refresh Token
 - Senden an den Token Endpoint führt zu einem neuen Access Token und einem neuen Refresh Token

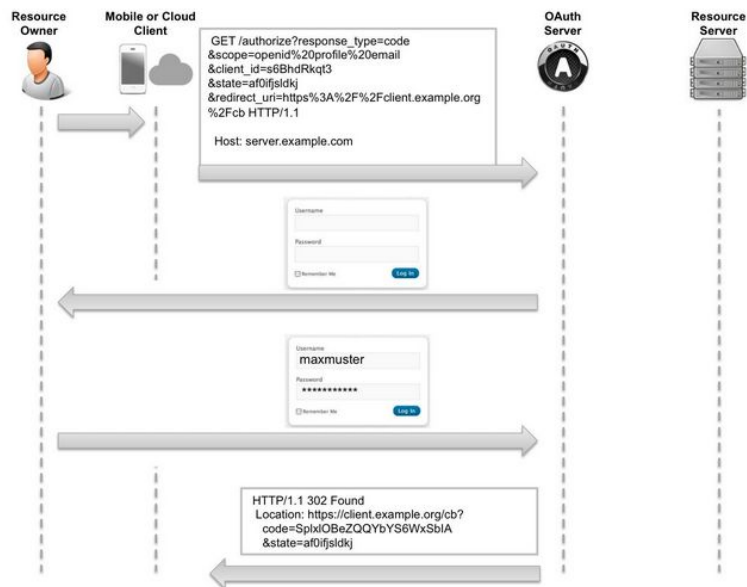
- Konzipiert für Clients, die sich keine relevanten Informationen dauerhaft speichern können oder sollen bzw. für die es auch keine individuelle Client-ID gibt
 - Client Secret
 - Refresh Token
- Beispiel
 - Single Page Application im Browser
- Es wird nur der Authorization Endpoint benutzt
 - Rückgabe ist bereits das Access Token
 - Token Endpoint wird nicht benutzt

- “Temporär” werden die Credentials des Resource Owners an die Client-Application übergeben
 - Die Anmelde-Logik ist Bestandteil des Clients
 - Klassische Welt der Authentifizierung an eine Anwendung!
 - “Temporär” ist eine Vereinbarung, keine Verpflichtung
 - Resource Owner muss dem Client vertrauen
- Client muss ClientID und Client Secret vorhalten
- ClientID + Client Secret + Credentials (Resource Owner) werden zum Token Endpoint gesendet
 - Rückgabe Access Token und Refresh Token
- Ab da: Alles wie beim Authorization Code Flow

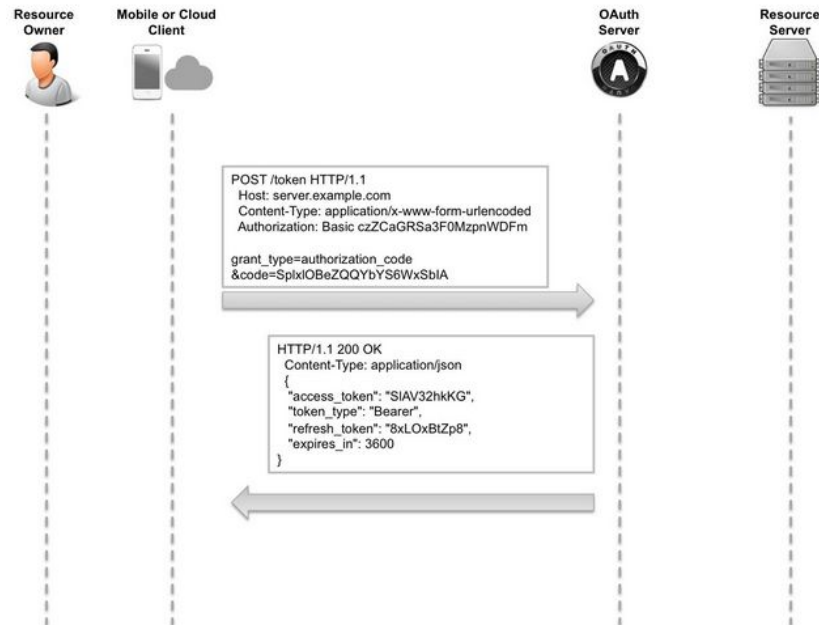
- Resource Owner “ist” der Client
- ClientID und das Client Secret werden zum Token Endpoint gesendet
 - Access Token + Refresh Token
- Ab da: Wie vorher...

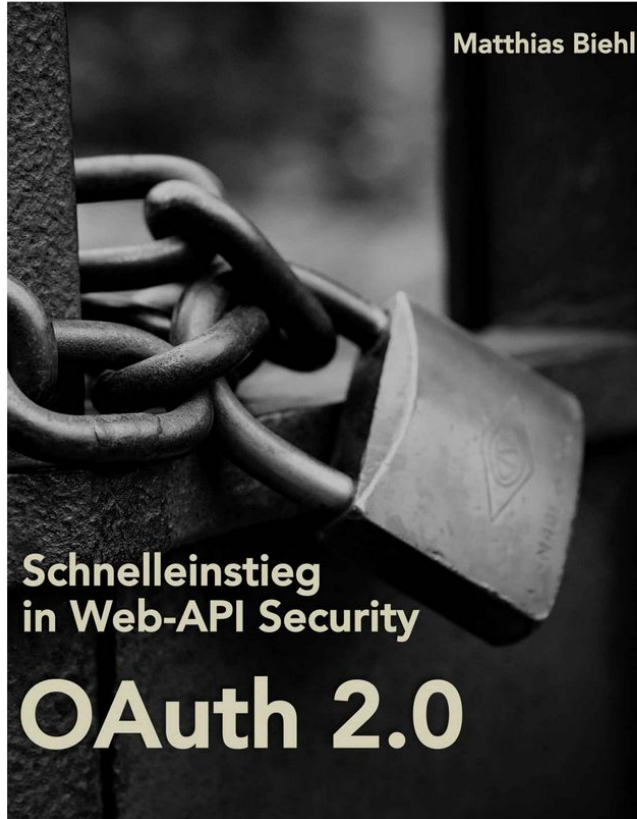
- Visualisieren Sie die beteiligten Rollen und den Ablauf des Authorization Code Flows
 - Beteiligte Rollen
 - Resource Owner
 - OAuth Provider
 - Client
 - OAuth Provider Admin
 - Beteiligte Endpoints
 - Welcher wird wann von wem aufgerufen?
- Melden Sie sich doch mal in ihrem internen Techem-Portal an
 - “Beobachten” Sie die Address-Zeile
 - Entwickler-Konsole des Browser sollte bei der Netzwerk-Darstellung insbesondere den Redirect mit dem Authorization Code zeigen!

Authorization Endpoint



TokenEndpoint





OAuth 2.0

Schnelleinstieg in Web-API Security

API University Series

www.api-university.com

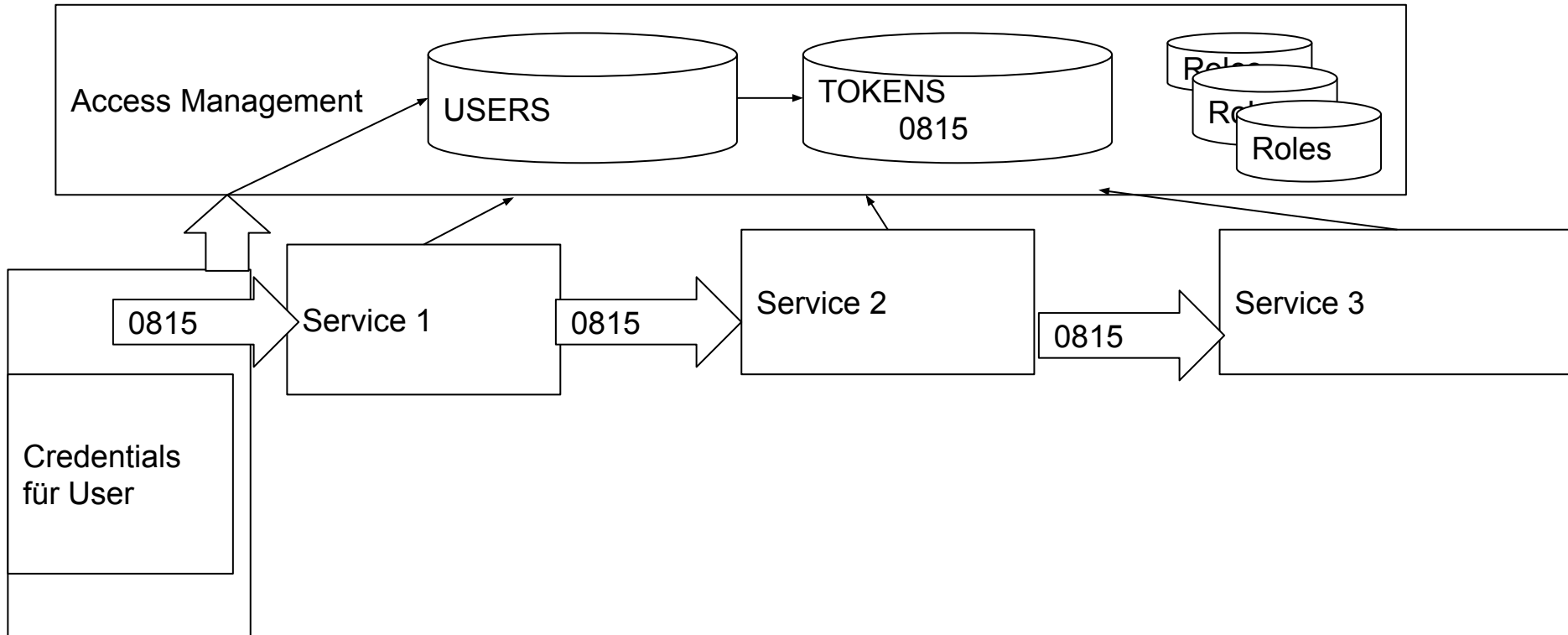
© Copyright 2014 by [Matthias Biehl](#)



OpenID Connect

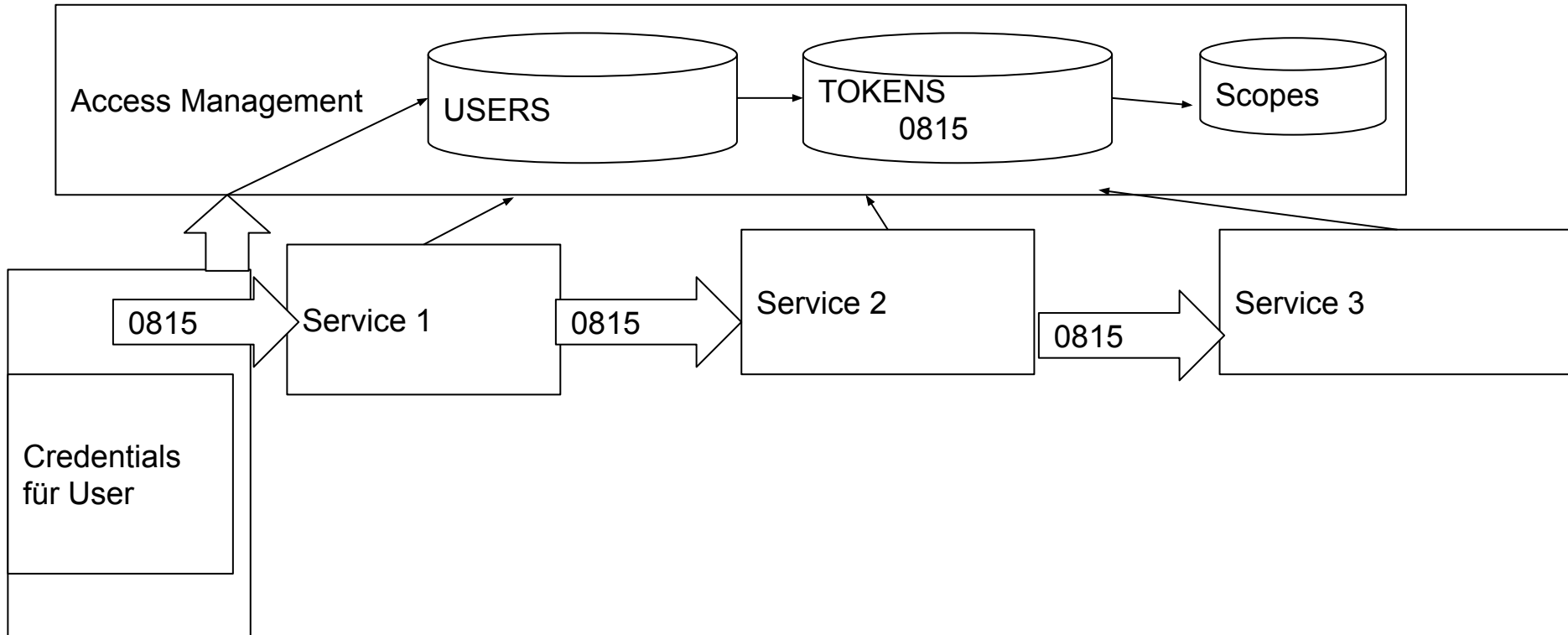
- OpenID Connect
 - Zugriff auf User-Informationen
 - UserInfo-Endpoint
 - Anmeldung unter Verwendung einer ganz anderen Web Seite
 - Registrierung an einer neuen Anwendung mit “Anmelden mit Google|Microsoft|GitHub”
 - “Identity as a Service (IAAS)”
 - Von Reference Tokens zu JSON Web Tokens
 - Zusammenhang zu OAuth2

Service-orientierte Systemlandschaft mit API Gateway



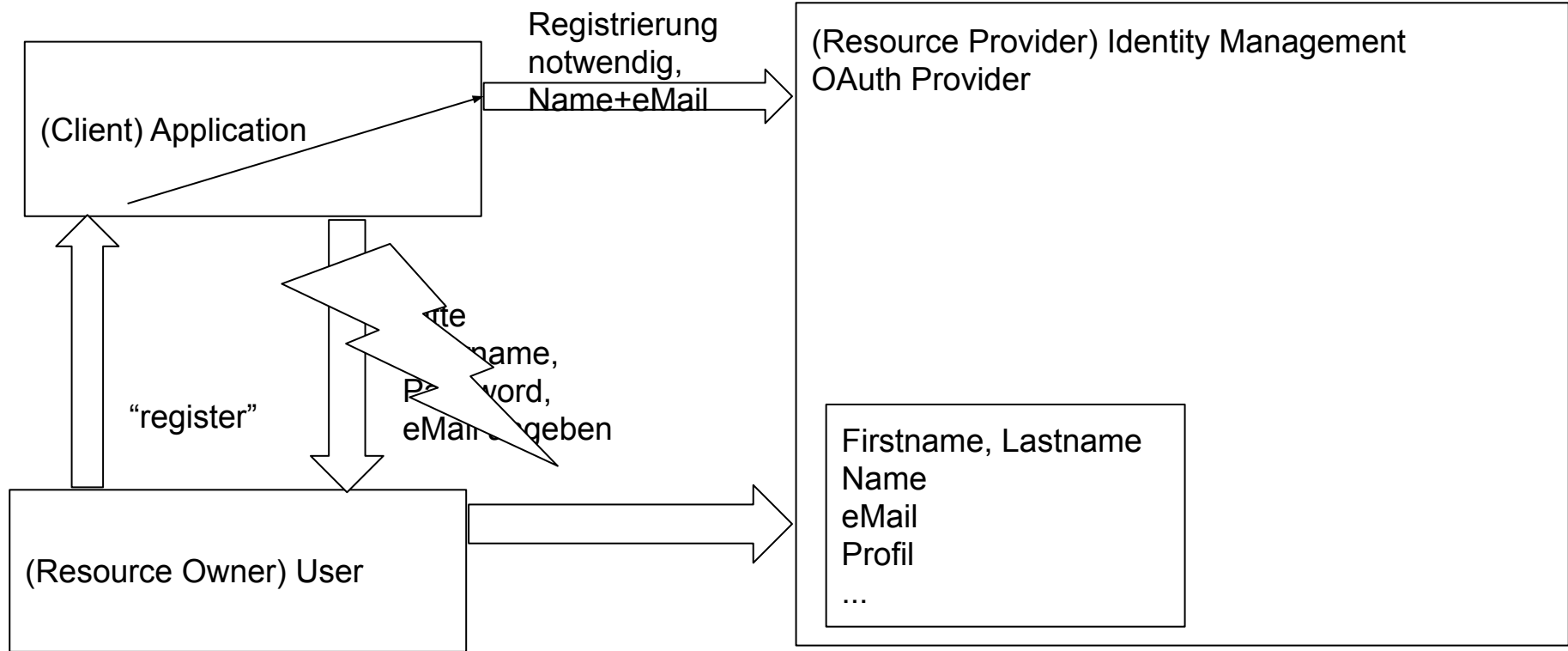
- Bei der Authentifizierung wird der Resource Owner bisher “nur” seine Credentials übermitteln
 - Keinerlei Kontrolle über die zu vergebenden Berechtigungen
 - Idee: Bereits bei der Authentifizierung kann zur Erstellung des Tokens ein Satz von “Rollen” = Scopes angegeben werden
- Scopes werden vom OAuth Provider Admin verwaltet
 - Eventuell die Mappings auf Service-Rollen
- Bestandteil von OAuth2

Service-orientierte Systemlandschaft mit API Gateway

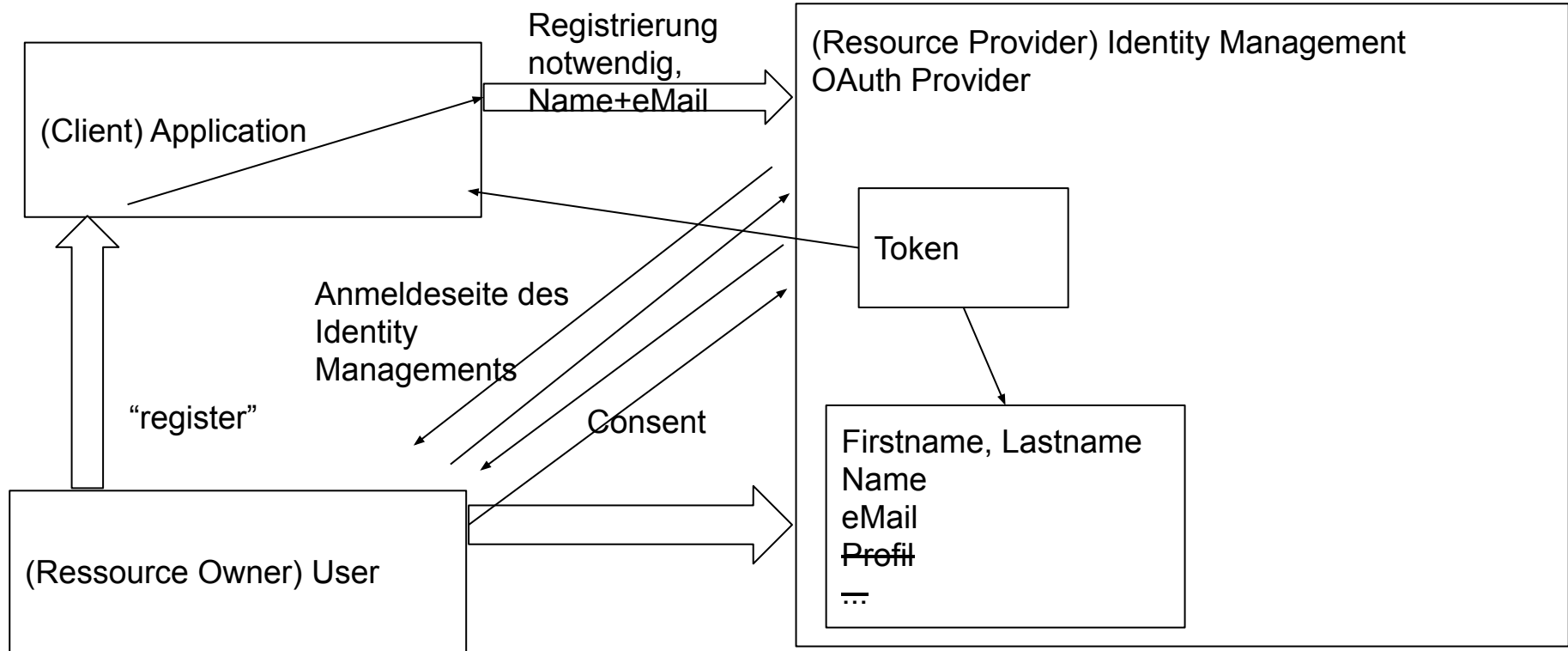


- Bisher: “Melde dich an!”
- Jetzt: “Die Anwendung XYZ möchte von Ihnen die folgenden Berechtigungen (Scopes) erhalten. Hierzu müssen sich sich authentifizieren”
- Neuer Schritt: “Consent”: “Sie werden nun die folgenden Scopes für die Anwendung berechtigen: ”

“Admin-less” OAuth Provider für “bestimmte Abläufe”



“Admin-less” Registrierung bei App



Woher weiß die Application vom Identity Management des Users?

- “Erraten”
 - Google?
 - Facebook?
 - Microsoft?
 - Yahoo?
 - ...

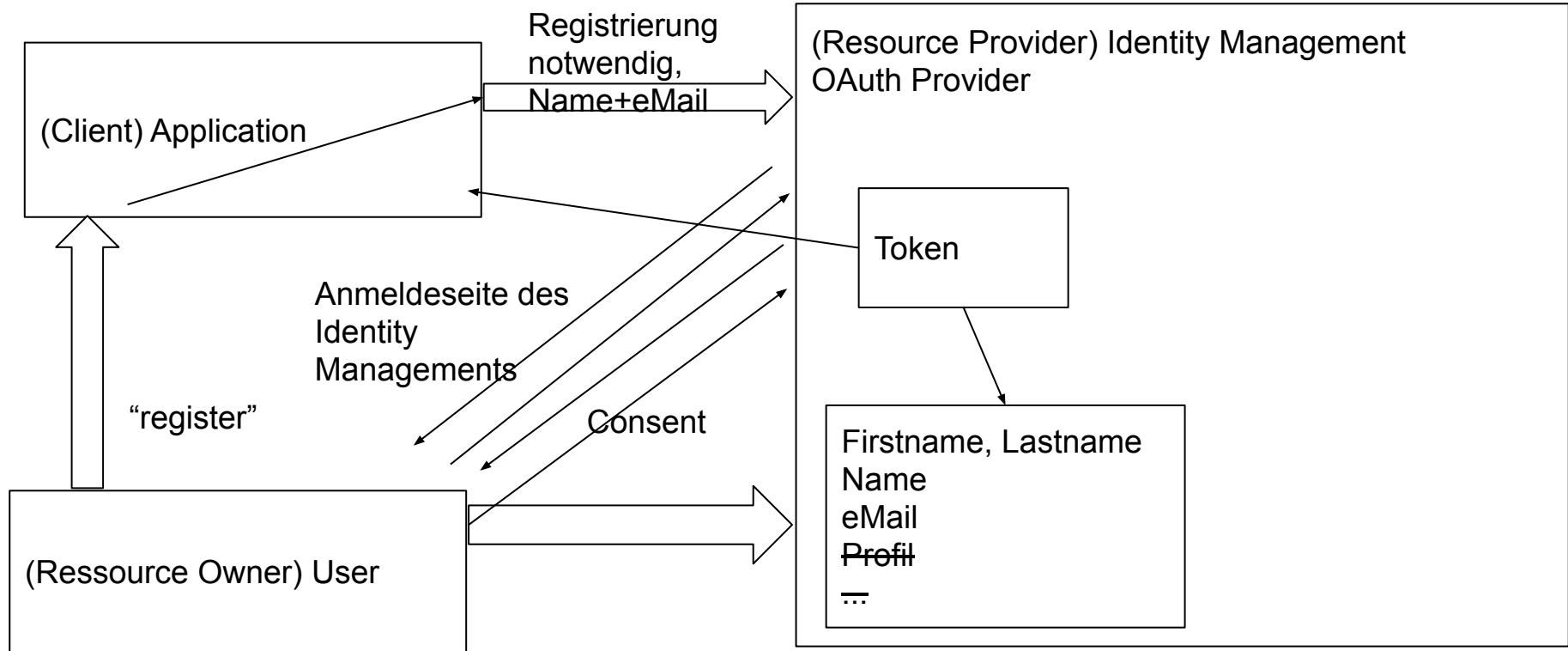
- Users
 - Verwaltet zentral seine Identität
- Application
 - Es muss kein eigenes Identity Management implementiert werden
- Provider eines öffentlichen Identity Managements
 - Benutzer-Bindung

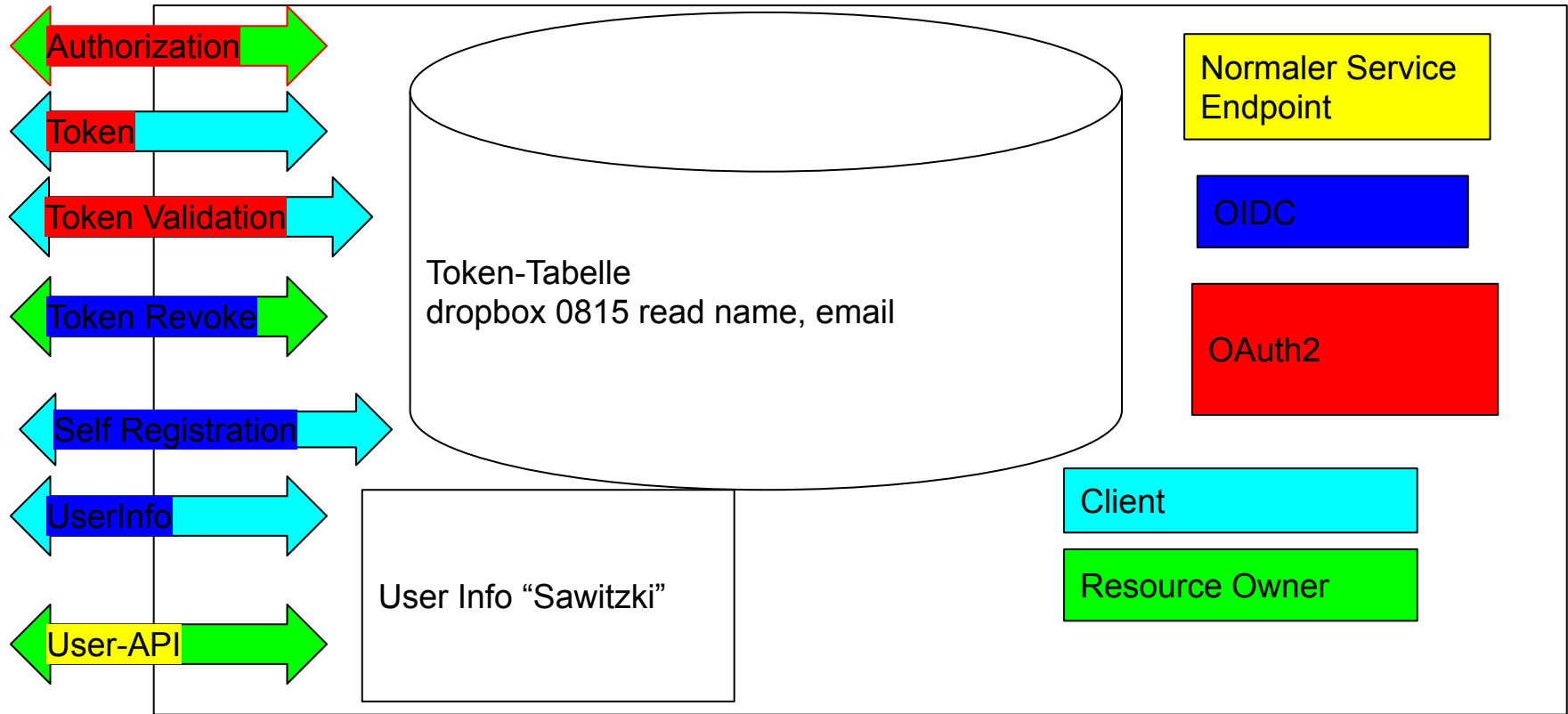
- Basiert natürlich auf OAuth2
- OAuth2 Scopes werden ergänzt um “Claims”
 - Claims regeln den Zugriff auf die Identity-Informationen
 - Claim: Name
 - Claim: eMail

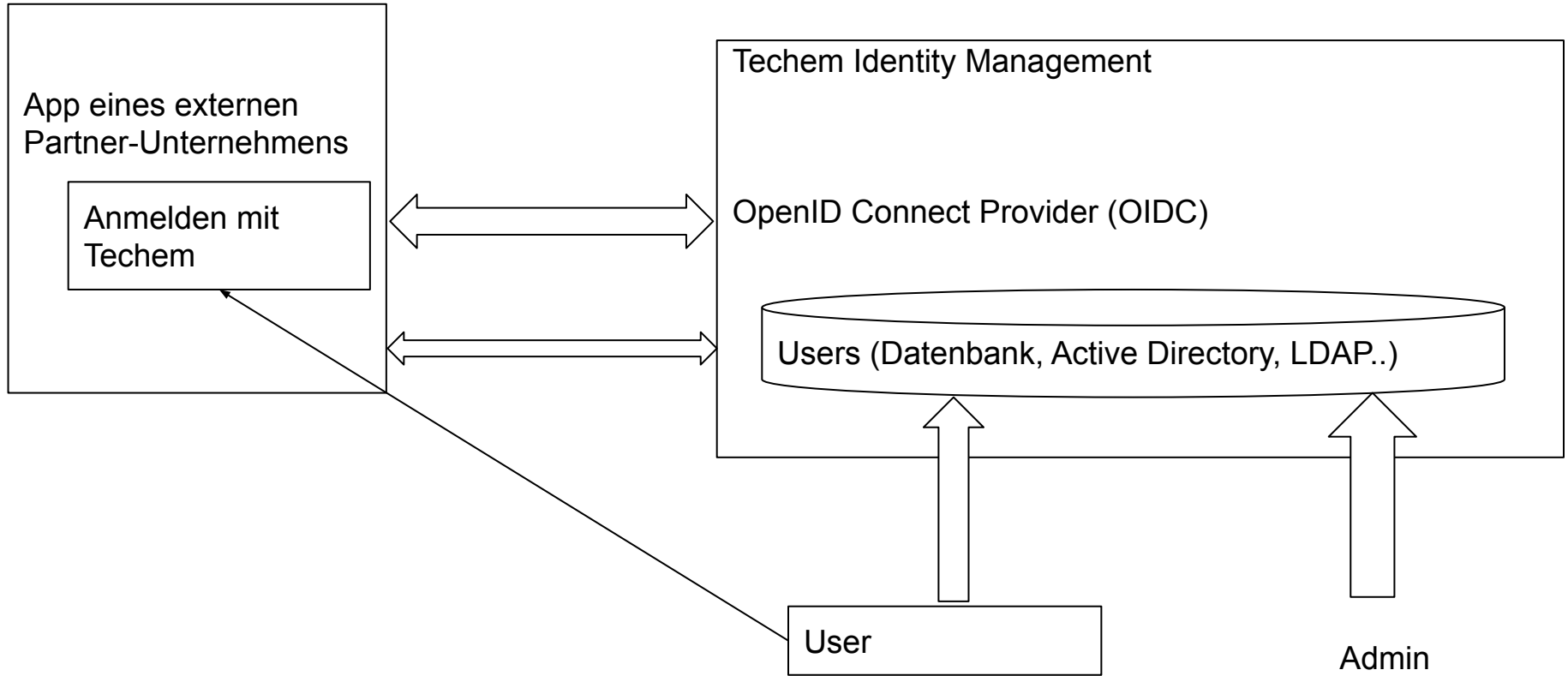
ToDo: OpenID Connect: Neue Endpoints?

- Hinweis: Mindestens 3 neue Endpoints sind notwendig!

“Admin-less” Registrierung bei App

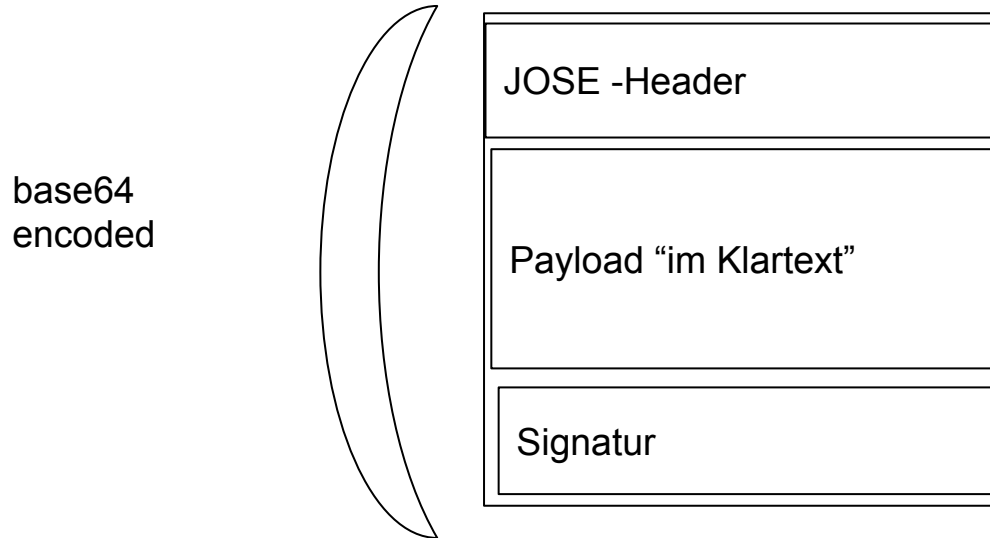






- Bisher genügen Reference Tokens
 - Validierung, “Was bedeutet dieses Token eigentlich” kann nur der OAuth Provider feststellen
 - Besser, intuitiver, einfacher: Value Token
 - Informationsträger
 - Token wird damit zu einem “Ticket”
- “Reference”
Für welche Aktionen
legitimiert (Klartext)
Digest/Signatur
- Änderungen im “Klartext” führen zu einer Invalidierung des Token, die jeder feststellen kann

- Benutzt JSON Web Tokens (oder SAML)
 - “JWT”, “Tschott”
 - Spezifikation der jwt.io



- Access Token als JWT
- Refresh Token als JWT
- Authorization Code bleibt im Wesentlichen eine Zufallszahl
- Neu: ID-Token
 - Token Endpoint liefert Access & Refresh Token und optional auch gleich ein ID-Token
 - Entspricht dem User Info

Beispiel

Web Frontend

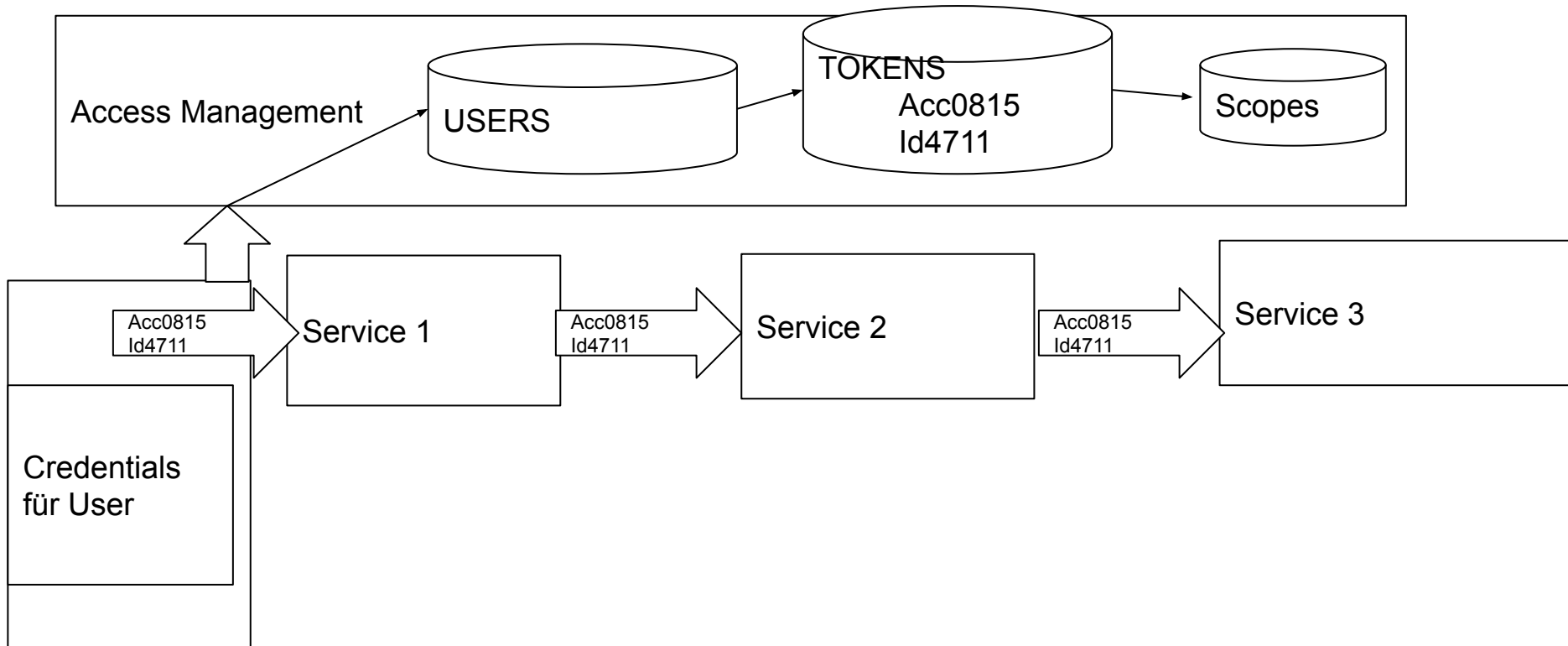
- Wechseln in das Verzeichnis angular_sample
- npm install
 - Download-Orgie
- npm start
 - CHECK: localhost:4200 als Einstieg

- Es funktioniert!
 - Entwickler-Werkzeuge im Chrome-Browser
 - Aufruf des Redirect-Endpoints nach erfolgreicher Authentifizierung
- Schauen Sie sich die verwendeten Aufrufe etc. an
- JWT in jwt.io decoden lassen
- “Play around...”

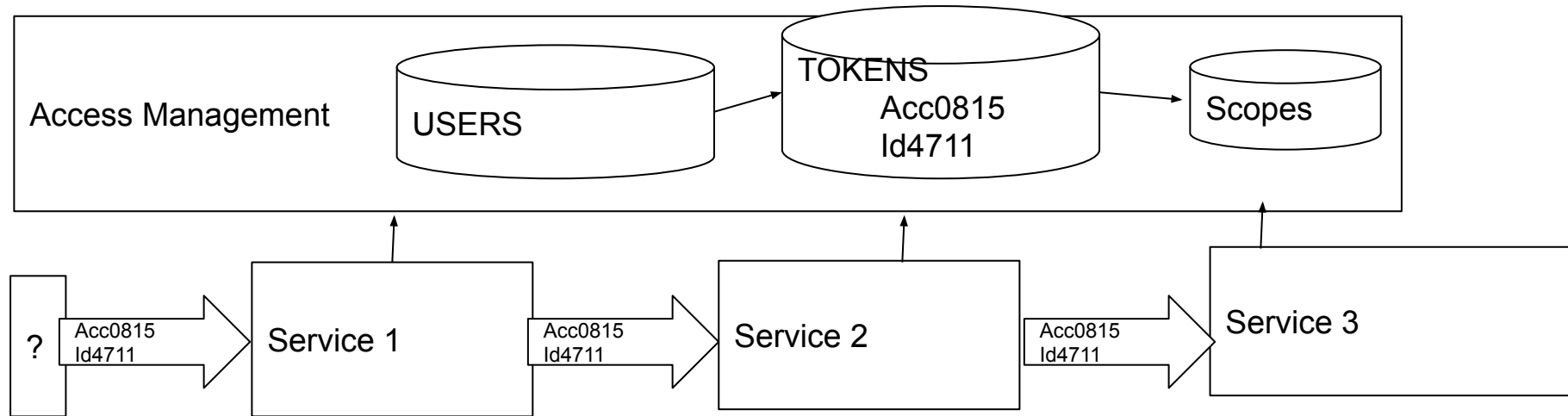
Services

- Was ist der geeignete Flow?
- OAuth oder OIDC mit JWT
- Wie geht es weiter mit den Tokens?

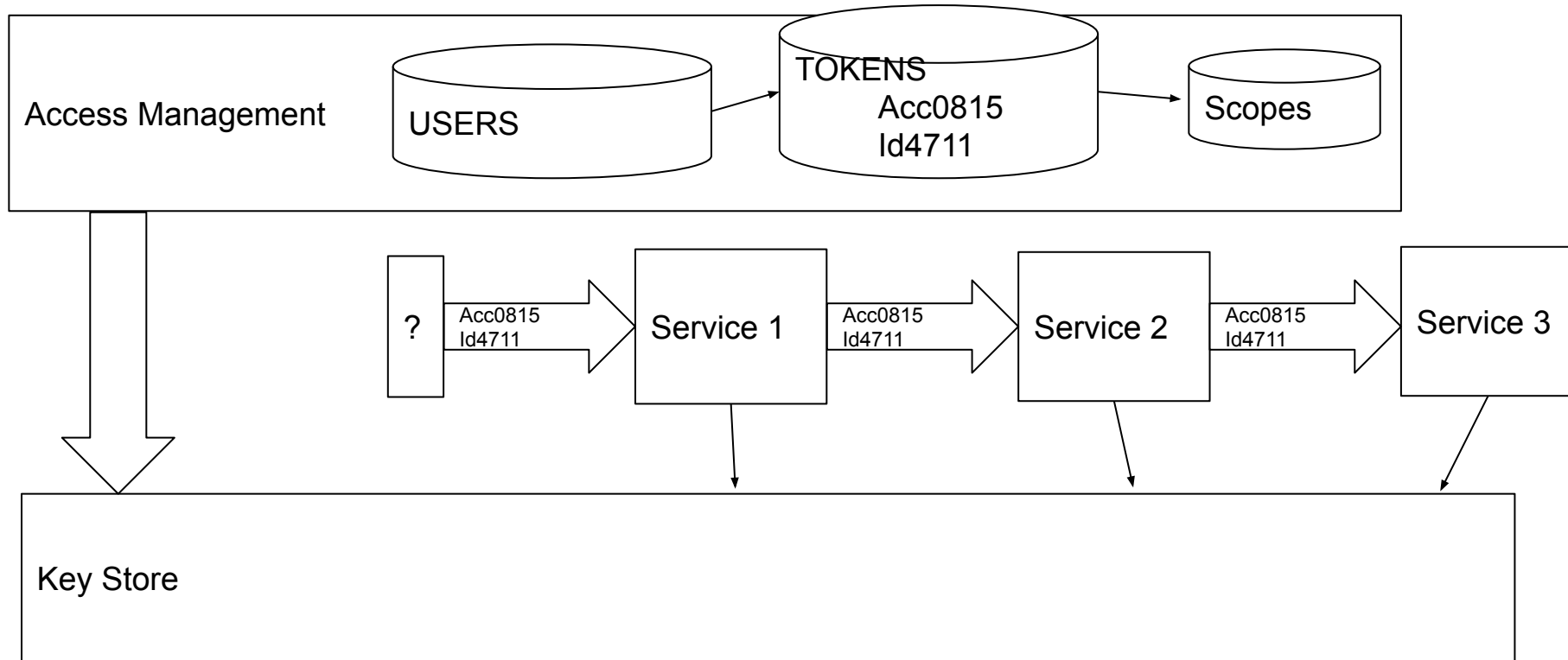
Service-orientierte Systemlandschaft mit API Gateway



Service-orientierte Systemlandschaft ohne API Gateway



Validierung des Tokens gegen KeyStore

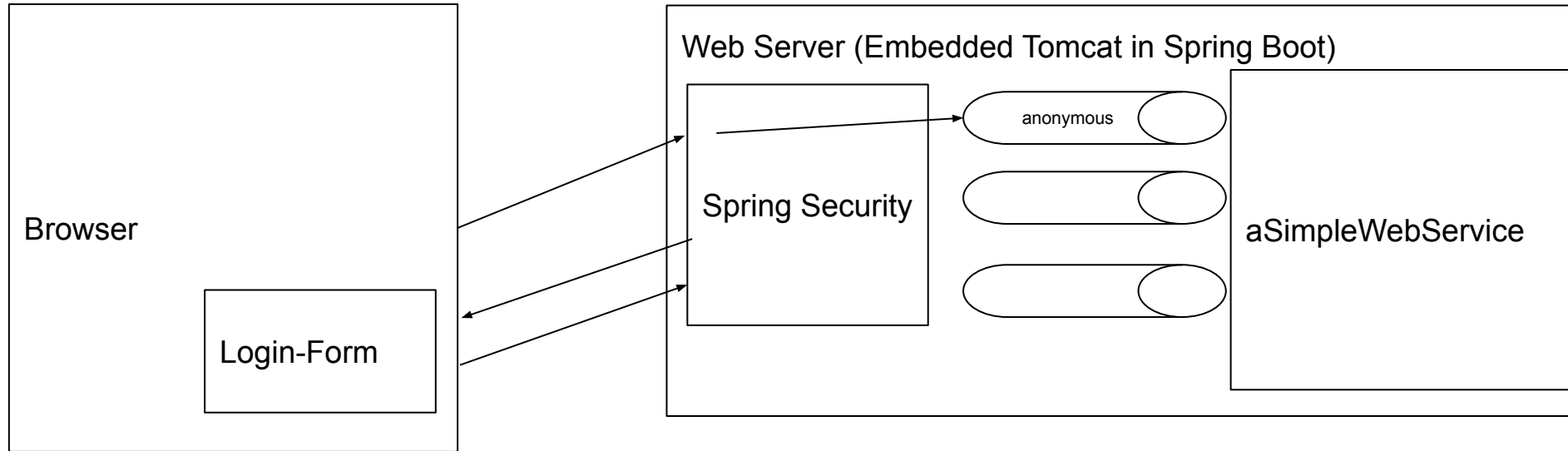


Umsetzung mit Spring Security

- RESTful Webservice abgesichert mit OIDC
- Realisierung mit Spring Boot und Spring Security
- Analog: Web Anwendung
-

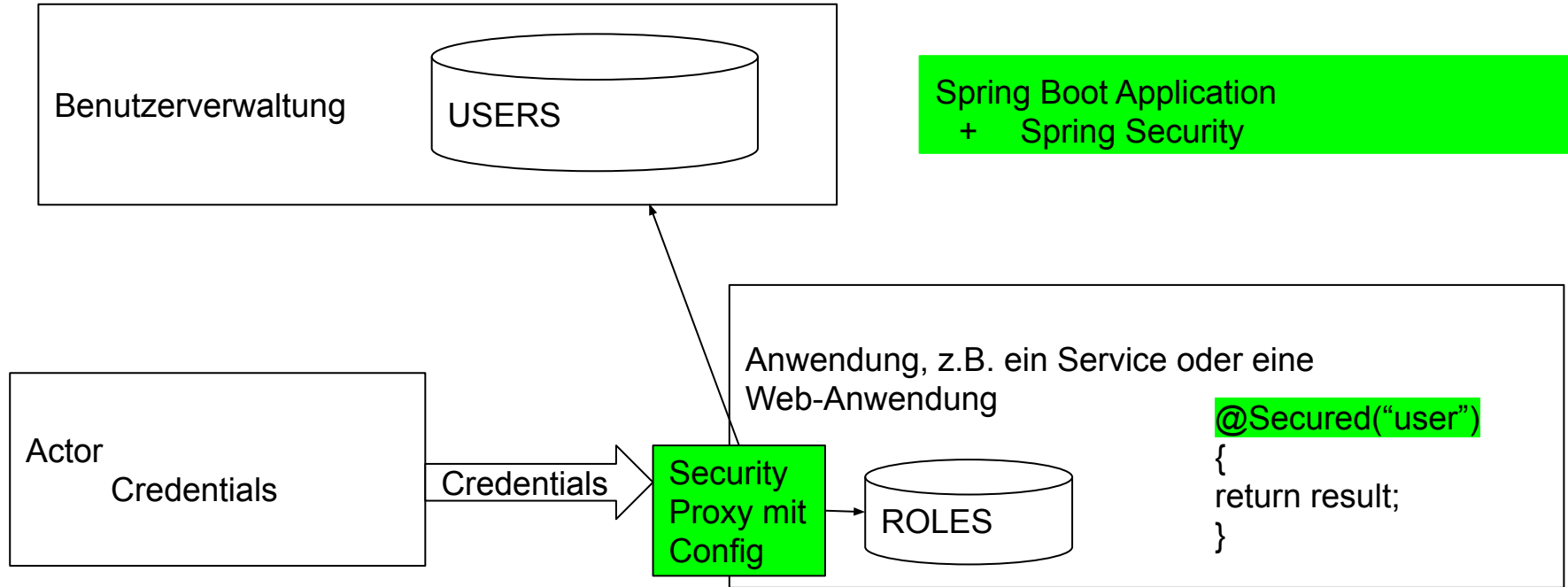
- Spring Boot-basierte Applikation
 - Entwicklungsumgebung
 - Eclipse (STS), IntelliJ
 - Aktueller Projektstand
 - <https://github.com/Javacream/org.javacream.training.spring.security/tree/046d8aceec8a74770ce7d7dce260f32b84fe90a9>
 - Ein simpler Web Service ohne jegliche Absicherung...

- Weiteres Spring Projekt: Spring Security
 - Wie üblich: Starter vorhanden
 - Im pom.xml eintragen
- Auto-Konfiguration (“Spring Boot weiß schon was sie wollen und brauchen”) funktioniert hier nicht komplett nahtlos
 - Eigene @Configuration-Klasse
 - Definiert die abgesicherten Endpoints über ein URL-Mapping
- Mock für unser Access Management



- <https://github.com/Javacream/org.javacream.training.spring.security/tree/99d163446610a7a7c7786632c2de81ea16794e1a>

Bestandteile einer aktuellen, etablierten Lösung



- <https://github.com/Javacream/org.javacream.training.spring.security/tree/09d6342fa14060edc46807614cc33872587f09d2>
-

- JSON Web Token ist eine völlig unabhängige Spezifikation!
- Implementierung im Java-Umfeld, JWT-Library
 - Erzeugen, Validieren, Encodieren, Decode
- Erzeugen eines JWT-Tokens: Eigener, simpler JWT-Service
 - `authorize(user, pwd): JwtToken`

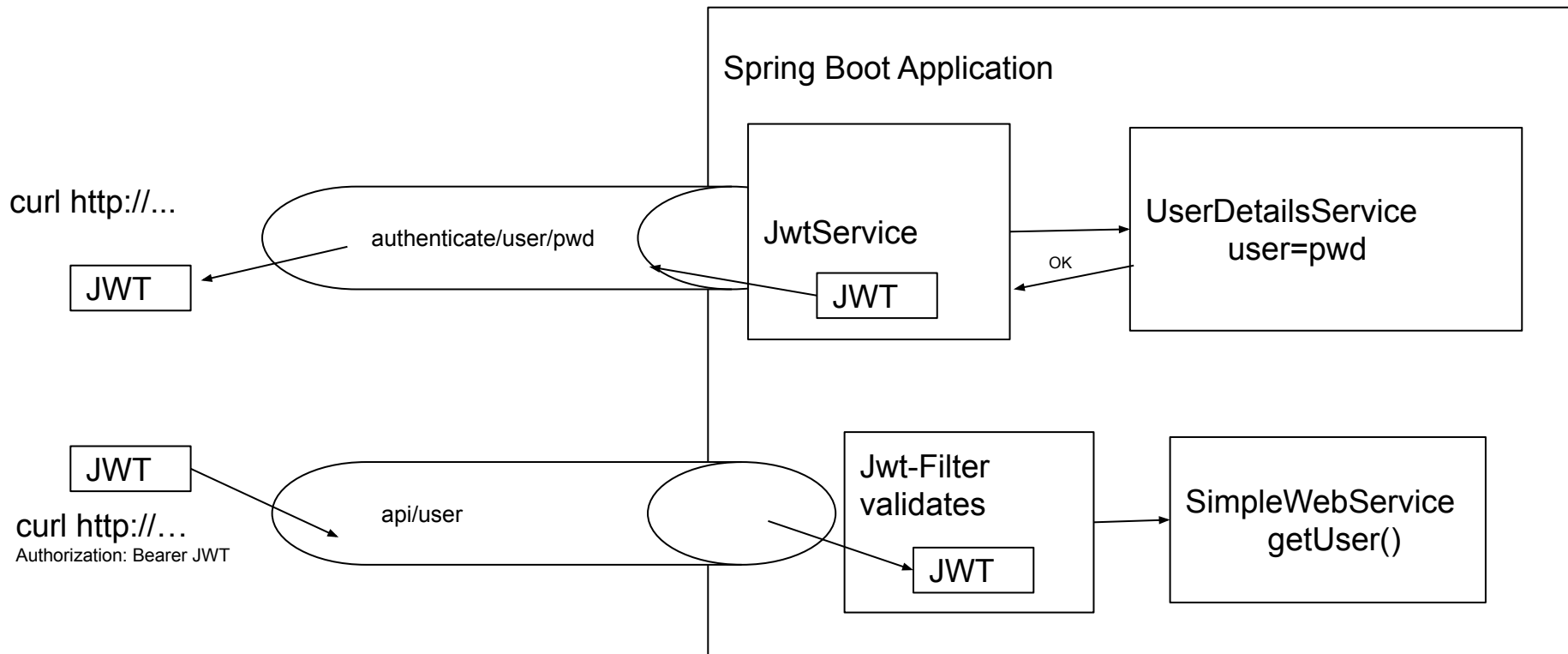
- Dependency auf JWT-Bibliothek im POM
- JwtUtil zur Vereinfachung
- Erweiterung der Konfiguration
 - AuthenticationManager wird “injectable” gemacht
- JwtService
 - authenticate als REST-Endpoint
 - Dependency auf UserDetailsService, AuthenticationManager

- <https://github.com/Javacream/org.javacream.training.spring.security/tree/a9fa75ae731ba6d9b26abeba047fb0da2ad5d672>

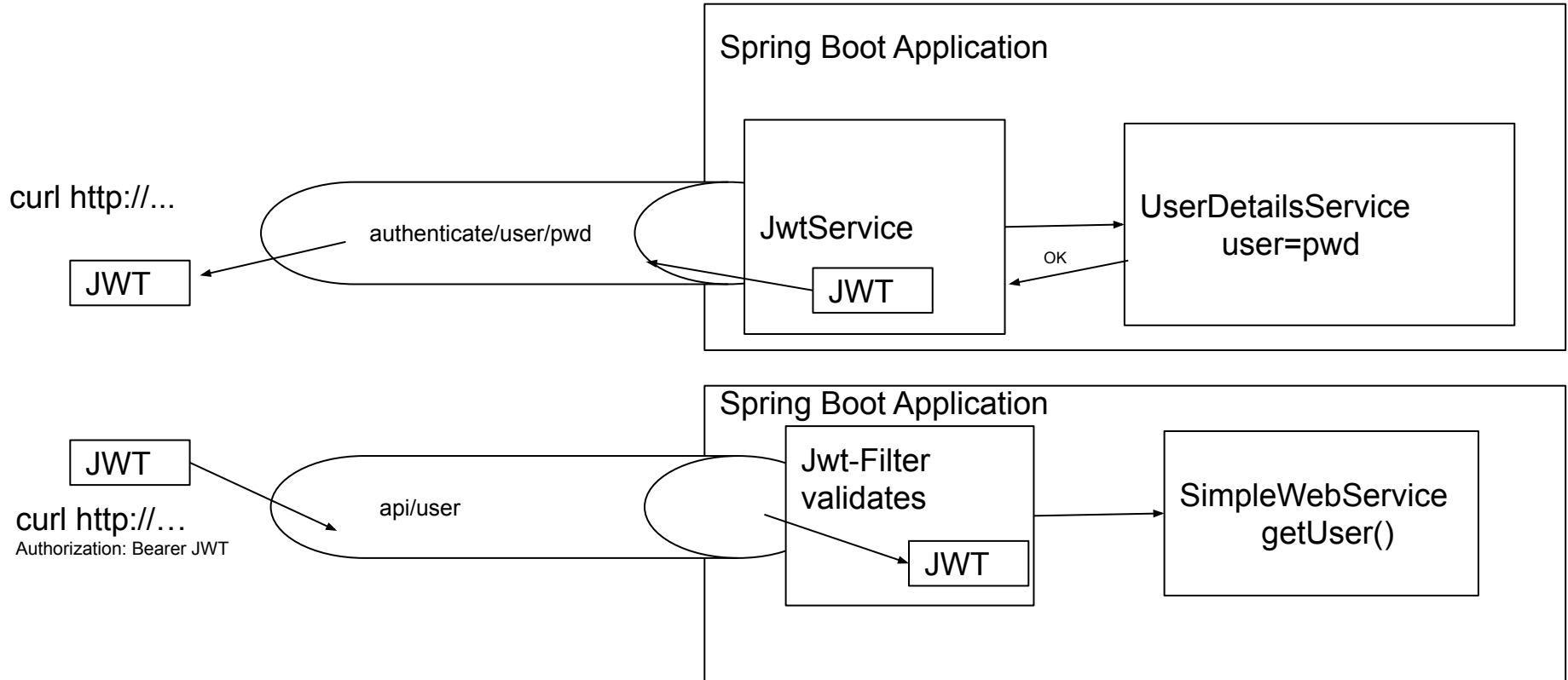
- Umsetzung erfolgt über einen normalen Servlet-Filter
 - Liest den Authorization-Header
 - Validiert das Token
 - Bestimmt und setzt die Rollen
 - Rollen sind nicht Bestandteil des hier implementierten JWT
- “Authorization Flow” über eine Sequenz von curl-Aufrufen

- <https://github.com/Javacream/org.javacream.training.spring.security/tree/748fbbe7c72789bcf6b4e7f79a90f9eba3e329a9>

Die eigene Lösung in der Übersicht



Sinnvoller, aber aufwändiger wäre:



- <https://github.com/Javacream/org.javacream.training.spring.security/tree/8e7e10db1b44cd9bb356c2783d76e43284a0e791>