Security und Spring Boot

Vorstellungsrunde

- Name und Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Konkrete Problemstellung
- Individuelle Zielsetzung

Einführung

Common Sense

Bedrohungsszenarien

- Software-Anwendungen sind anfällig gegen Hacker-Angriffe
- Schäden durch erfolgreiche Angriffe können immens sein
  - Konkreter Schaden
  - Prestige- und Image-Verlust
- Konsequenz: Es ist obligatorisch, Security-Lücken zu erkennen und zu beheben

Typischen Abwehrmaßnahmen

integrata cegos

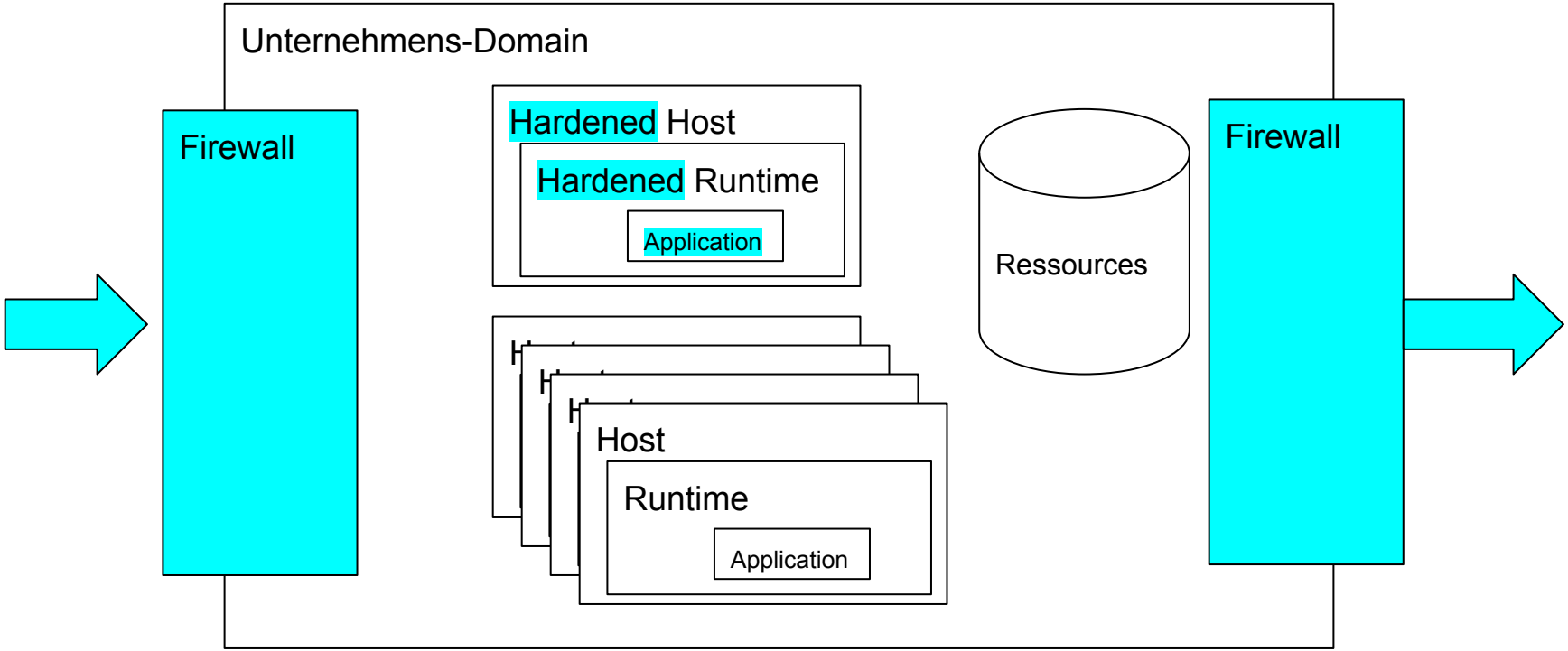
- Sichere Kommunikation durch Verschlüsselung
  - https
  - VPN (Virtual Private Network)
- Firewall
  - Abschottung des internen Netzwerks von unzulässigen Zugriffen von Außen
    - Port Scans
    - Denial of Service
    - Injections, z.B. Sql-Injection
- Hardening der Umgebung
  - Betriebssystem mit einer Einführung von Zugriffsbeschränkung
    - Prinzip der “Least Privileges”
  - Runtime, z.B. Konfiguration eines Applikationsservers

Abwehrmaßnahmen: Anwendung

- Authentifizierung
  - Benutzerverwaltung mit Benutzer/Password, Credentials
- Autorisierung
  - Anwendungs-Rollen mit Berechtigung
- Daten-Validierung bei jeglichem Transfer von Informationen über Domänen-Grenzen hinweg

Gesamtbild Security

integrata cegos



Praxis

Schutz vor Security-Angriffen

integrata cegos

- 100%iger Schutz ist unmöglich
  - z.B. korrumpierter Administrator
  - Gehärtete Umgebungen haben immer irgendwelche noch nicht erkannten Lücken
  - Anwendungen sind ebenfalls nie fehlerfrei
    - insbesondere wenn eine agile Softwareentwicklung **missverstanden** wird
  - Verschlüsselungs-Algorithmen
    - Erfolgreicher Angriff
      - Naiver Versuch: Äonen
      - Organisierter Angriff: Masterarbeit an der Universität
      - Organisierter Angriff mit Backdoors: Quasi Echtzeit

Mängel am “Gesamtbild Security”

- Verantwortungs-PingPong
  - “Wer kann was” ist nicht “Common Sense”
- Konkrete Umsetzung sowohl von Verschlüsselung als auch von Authentifizierung verlangt den Einsatz eines Identity Management Systems
- Die meisten erfolgreichen Angriffe erfolgen **innerhalb** der Unternehmens-Domäne
  - Admin-Anwendungen
  - Unsichere Datenablage in Ressourcen

ToDos

Was ist alles zu tun?

integrata cegos

- Security in einer agilen Software-Entwicklung?
- Einführung eines Identity Managements
  - Kerberos, OAuth2
- Wie erfahre ich von Security-Problemen in
  - Betriebssystemen?
  - Runtimes?
  - Frameworks?
- Secure Build-Prozess
  - Automatisierte Tests und Scans
- Review- und Test-Strategien zum Erkennen von Vulnerabilities

Organisationen

MITRE

- Common Vulnerabilities Enumeration
  - Informationsbasis mit allgemeinen Strategien von Angreifern
  - cve.mitre.org
- Common Weakness Enumaration
  - Sammlung typischer Programmier-Fehler
  - cwe.mitre.org
- Att&ck
  - Tool zur Einschätzung der Gefährdung einer Anwendung

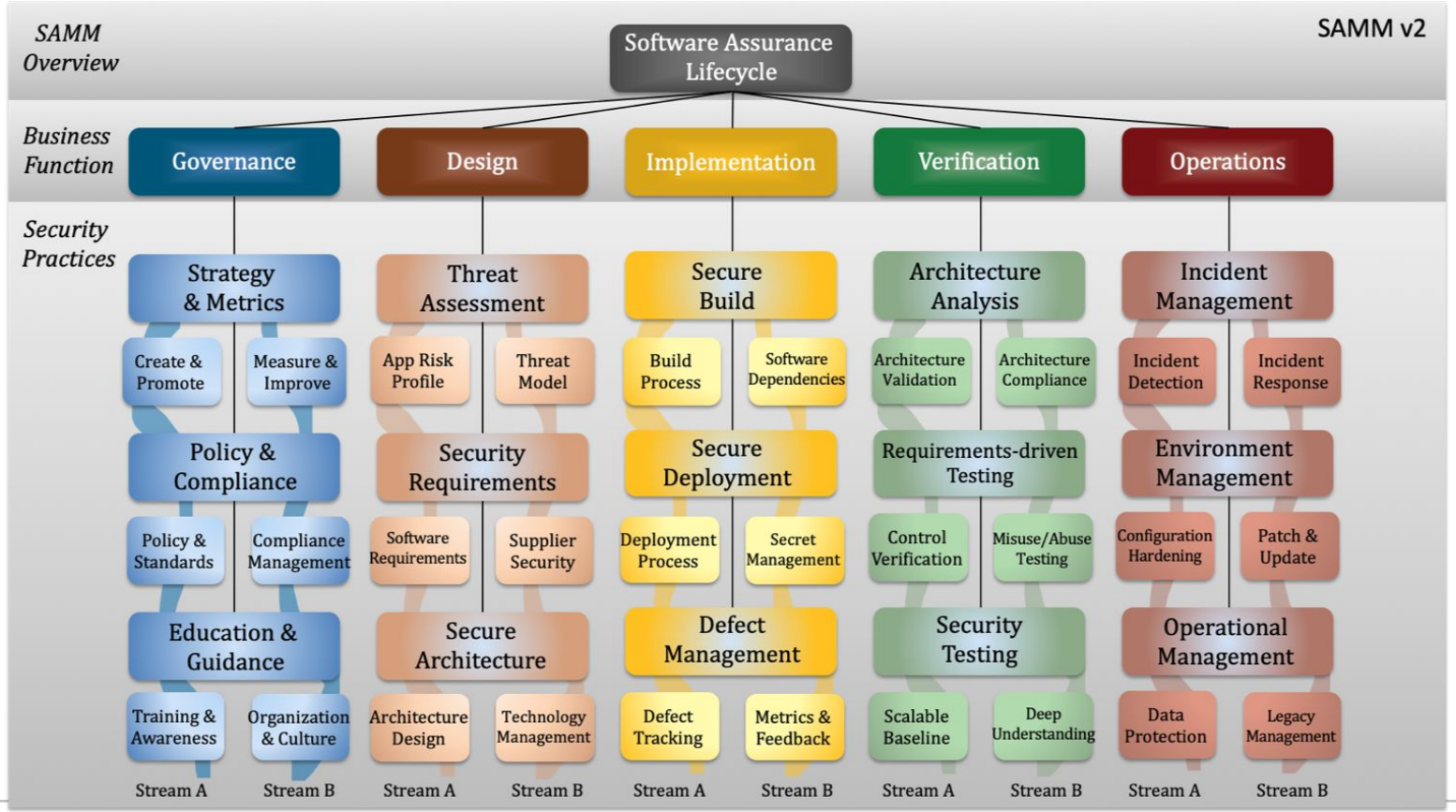
OWASP

integrata cegos

- Sehr bekannt über ihre OWASP Top 10
  - Liste der aktuellen Security-Lücken von Anwendungen
- Deutlich mehr als nur die Top 10!
  - Web Goat und Web Wolf
    - Trainings-Umgebung zur Security-Sensibilisierung von Entwicklern
  - ESAPI
    - Framework zur implementierung einer **W**eb **A**pplication **F**irewall
      - Primärfokus auf die Eingabe-Validierung
  - Security Assurance Maturity Model

OWASP SAMM (Security Assurance Maturity Model)

integrata cegos



OWASP Top 10

integrata cegos

- Rating
  - Kombination aus
    - Häufigkeit
    - Relevanz = Abschätzung des Risiko-Profils
- Aktualisierung alle paar Jahre
  - Aktuelle Version von 2021

Top 3: Injection

integrata cegos

- Die bekannteste ist “Sql-Injection”
  - Eine Information aus einer potenziell unsicheren Domäne wird als Skript-Befehl interpretiert
    - Eingabe: ‘ or 1=1
- Vorsicht
  - XML-Injection, JSON-Injection, …
  - Auch Log-Dateien können Injection-Befehle enthalten

Ansonsten

- die meisten Top 10s haben mit Fehlern in der Programmierung der Anwendung gar nichts zu tun

Praktikum

Anforderungen

- Java-Installation >= 8
- Java-Entwicklungsumgebung
- Freien Zugang auf das Internet zum Nachladen weiterer Java-Bibliotheken
-

Umsetzung

- Ihr eigener Arebitsplatzrechner
- Alternativ ein Remote-Rechner der Integrata-Cegos
  - tn02.raum01@integrata-cegos.de 33059_tn02 Sputaj
  - tn03.raum01@integrata-cegos.de 33059_tn03 Heyder
  - tn04.raum01@integrata-cegos.de 33059_tn04 Senar
  - tn05.raum01@integrata-cegos.de 33059_tn05 Döring
  - tn06.raum01@integrata-cegos.de 33059_tn06 Rainer
  - tn07.raum01@integrata-cegos.de 33059_tn07 wagner
  - tn08.raum01@integrata-cegos.de 33059_tn08 Franke
  - tn09.raum01@integrata-cegos.de 33059_tn09 Hellmann
  - tn10.raum01@integrata-cegos.de 33059_tn10 Schiffbahn
  - tn11.raum01@integrata-cegos.de 33059_tn11
  - tn12.raum01@integrata-cegos.de 33059_tn12
  - tn13.raum01@integrata-cegos.de 33059_tn13
  -

Beispiel-Anwendung

- Einfacher RESTful WebService
  - Zugriff über einen HttpClient möglich
    - Browser
    - cUrl
      - Windows z.B. über ein Git Bash
- Standalone executable Anwendung
  - .jar
- Web-Anwendung für ein Deployment z.B. in einem Apache Tomcat
  - .war

Einführen der Security: So nicht

integrata cegos

- Weitverbreitete These
  - “Jede Anwendung muss auch beim Versagen jeglicher Infrastruktur-Komponente (z.B. Firewall) immer noch sicher sein”
- Diskussion
  - Sawitzki: “Das ist völlig weltfremd”
  - Begründung
    - Angriff: Denial of Service im Programm erkennen?

Einführung Security

- Trigger
  - Begriff entstammt den Agile Notes von OWASP SAMM
  - Aufgabe ist die Entschlackung und damit Beschleunigung des Entwicklungsprozesses durch Einführung eines Regelwerks
- Beispiel
  - Trigger RESTful WebService
    - Gesetzt ist die Existenz einer Firewall sowie einer Runtime-Plattform
      - Definition einer Ausführungs-Domäne
      - Schutz gegen Angriffe wie Denial of Service
    - Trigger Java
      - Ausführung in einer Java Virtual Machine
        - Keine Probleme mit Buffer Overflows oder direkter Speicher-Manipulation
        - Security Updates (MITRE) und Incident Management

Trigger und Architektur-Entscheidungen

- Passt gut zusammen
  - Jegliche Architektur-Entscheidung kann als Security-Trigger dienen

Risiko-Analyse Part 1

- “Ein Benutzer kann Abläufe anstossen und Ergebnisse bekommen, die nicht für ihn/sie zulässig sind”
  - Authentifizierung und Autorisierung sind verpflichtend
    - Identity Management sowie ein Rollen-Konzept müssen eingeführt werden
    - Offene Flanke: Hierfür gibt es kein Standard-Konzept!
  - Unter der Voraussetzung “RESTful WebService mit Java” gibt es 2 aktuelle Frameworks zur Authentifizierung und Autorisierung
    - JEE (Enterprise Edition von Java)
    - Spring Security

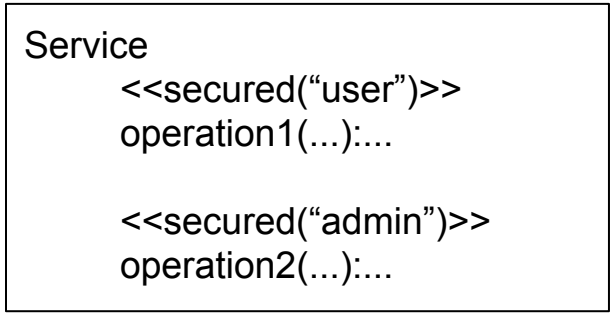
Beispiel: Spring Security

integrata cegos

- Aktuell das meist genutzte Security-Framework im Java-Umfeld
  - Kann ebenfalls problemlos von JEE-Applikationsservern benutzt werden
- Autorisierung
  - Methoden-Ebene (@Secured-Annotation)
  - Web Applikationen und auch RESTful WebServices über URL-Pfade
- Authentifizierung
  - Leichtgewichtige Test-Umgebung

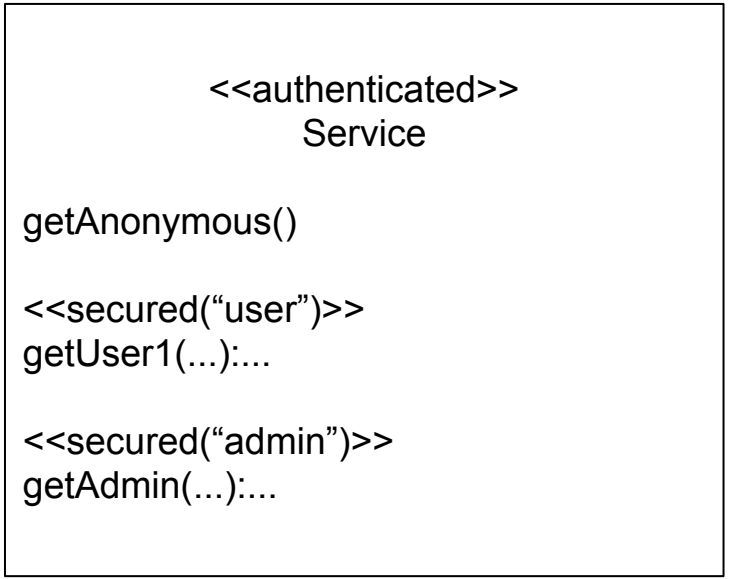
Das Rollenkonzept einer Anwendung

- Fachlich motivierte Umsetzung auf die technisch realisierten Prozesse/Methoden
  - WICHTIG: DAS ROLLENKONZEPT MUSS BEREITS VORHER GETESTET/REVIEWED WERDEN
  - User Stories enthalten das Rollenkonzept!
- Technisches Modell, z.B. mit einem Klassendiagramm



Die Beispielanwendung

integrata cegos



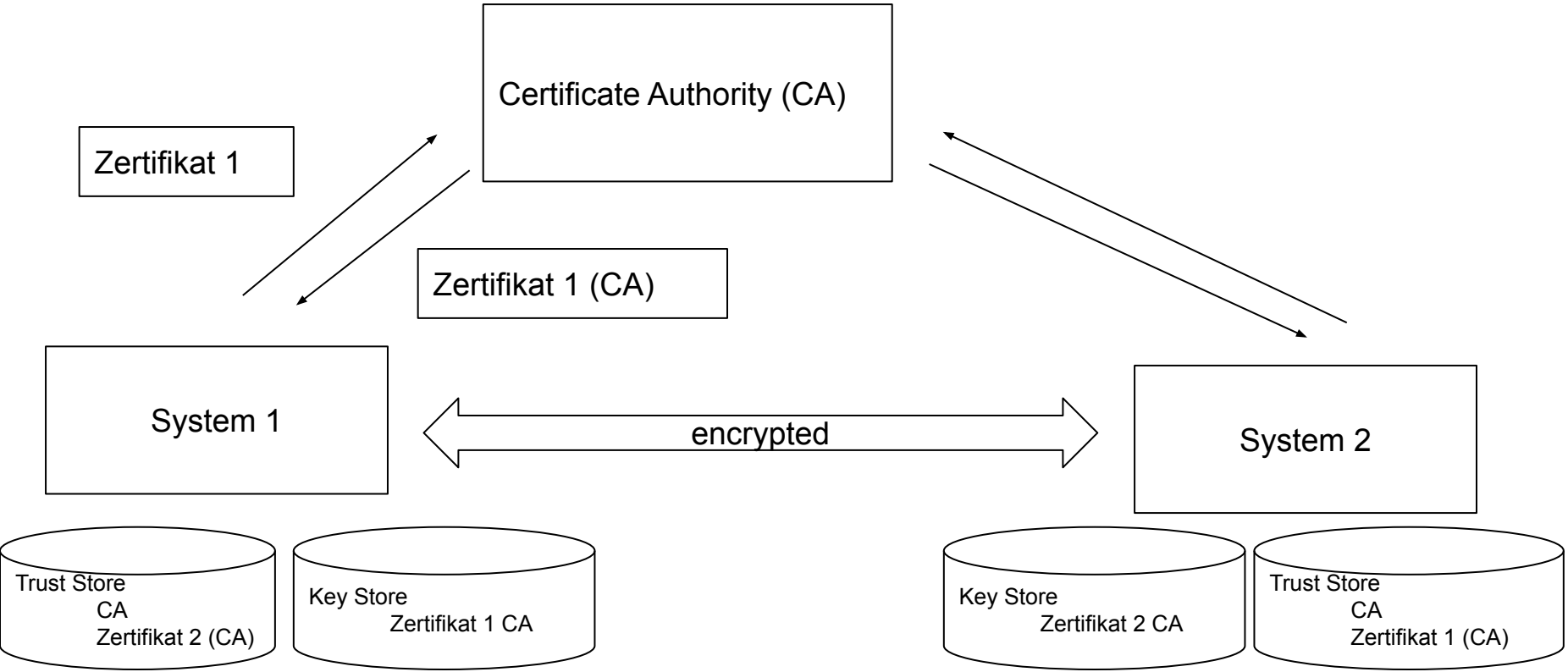
Identity Management

Verschiedene Optionen

- Username/Password-Übertragung
  - Verlangt notwendigerweise eine verschlüsseltes, abgesichertes Kommunikationsprotokoll
- Jedes beteiligte System wird sich mit einem Zertifikat ausweisen
  - Der Besitz des Zertifikats identifiziert den Benutzer eindeutig
  - Total klassisches System, jeder Personalausweis “funktioniert so”
- Token-basierte Systeme
  - Ein zentrales System verwaltet alle Benutzer und deren Credentials
  - Dieses System stellt Tokens aus, die für eine weitere Authentifizierung benutzt werden können
    - Tokens verhindern die Notwendigkeit der Übertragung von Security Credentials und können auch in vollkommen unsicheren Umgebungen benutzt werden

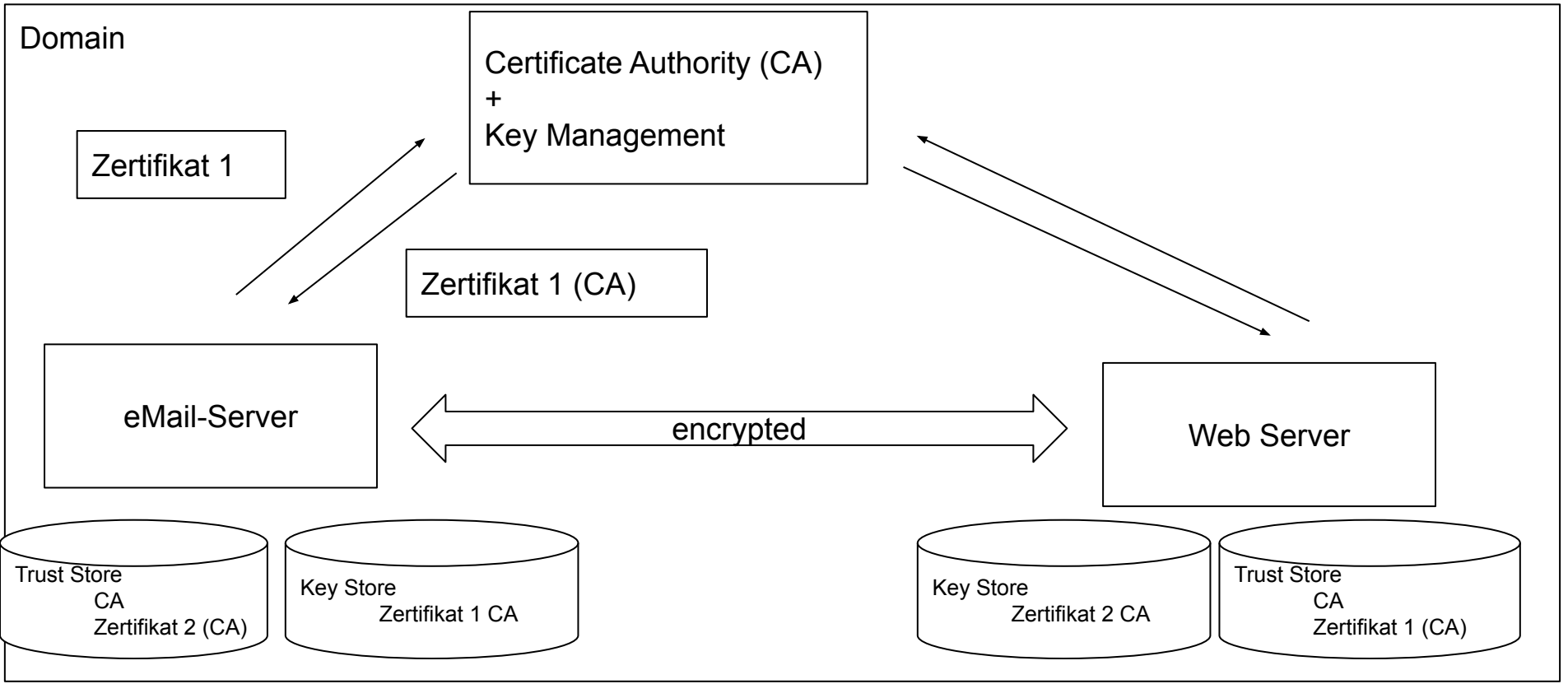
Secure Socket Layer (SSL)

integrata cegos



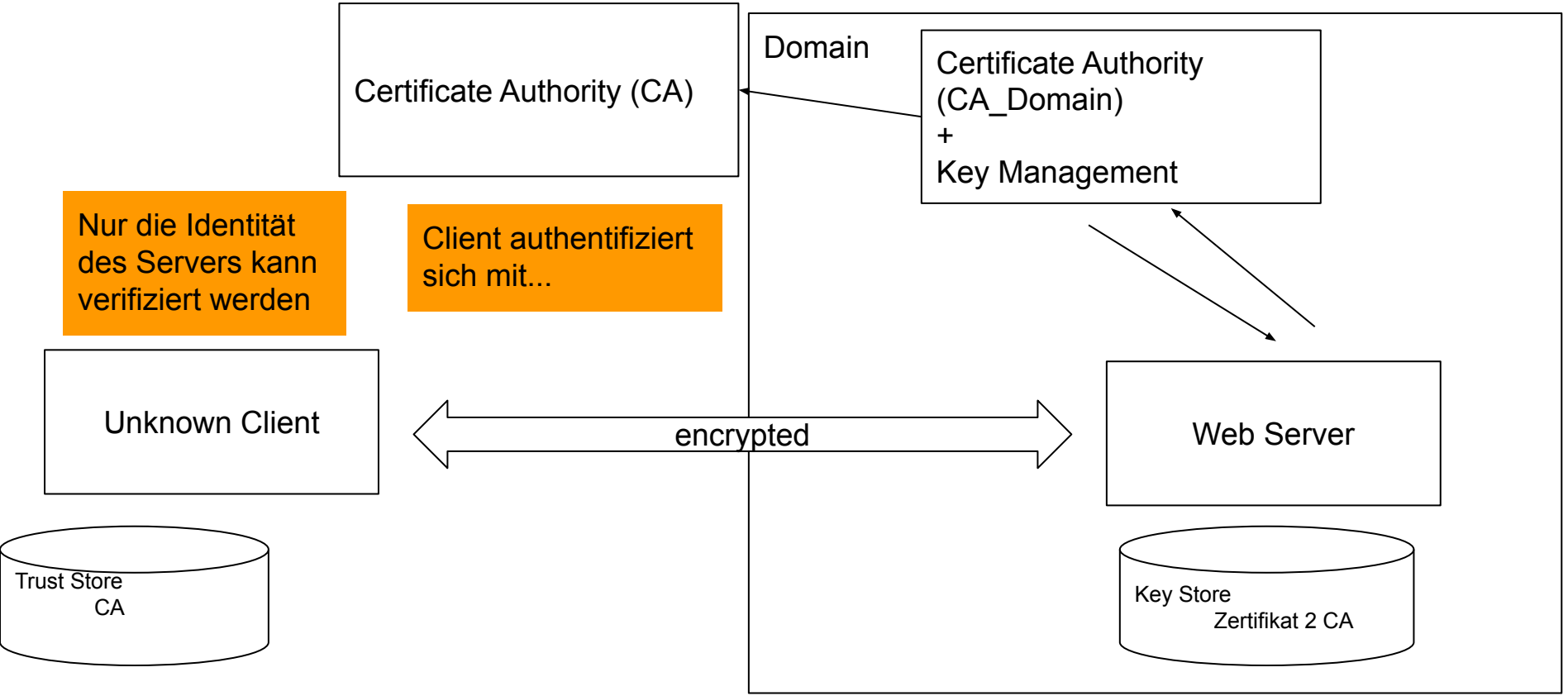
Systeme innerhalb einer Domäne

integrata cegos



Internet: Abgespecktes Identity Management

integrata cegos



Sichere Kommunikation

Sichere Datenübertragung

integrata cegos

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

Grundlegende kryptographische Algorithmen

integrata cegos

- 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

Digests

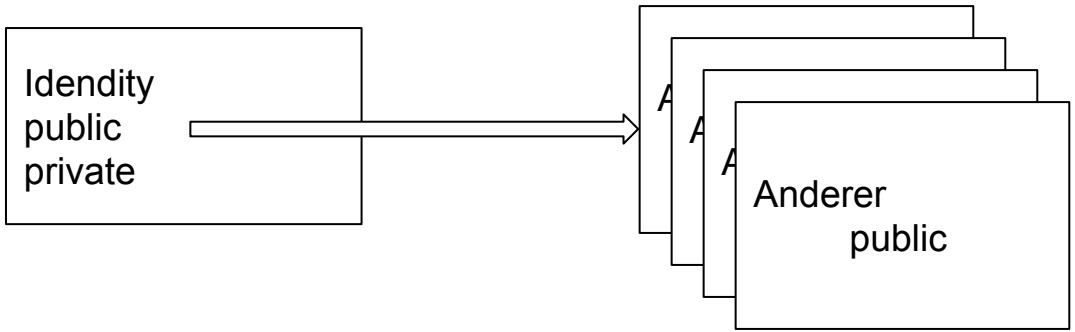
- Daten werden unidirektional in einen Hashwert überführt
  - “Unendlich” -> Endlich
  - Die Sicherheit des Algorithmus’ wird bestimmt z.B. über die Wahrscheinlichkeit von Kollisionen (unterschiedliche Eingangsdaten produzieren denselben Wert) bzw. “Mustern”, die auf die ursprünglichen Daten hinweisen
    - Kollisionen sind per se unvermeidbar

Symmetrische Verschlüsselung

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

Asymmetrische Verschlüsselung

- 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



Schlüsselmanagement

- 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, integere 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 Schlüsselgenerierung 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.

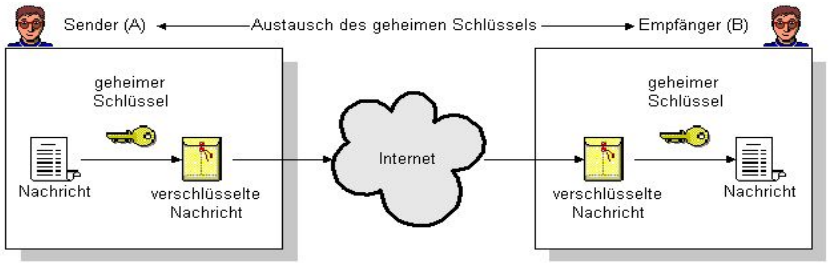
Schlüsseltrennung

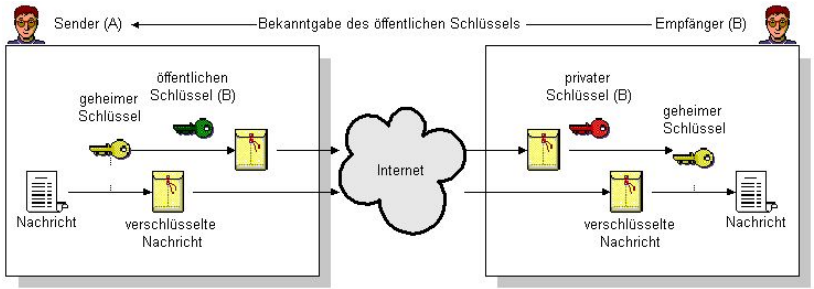
Kryptographische Schlüssel sollten möglichst nur für einen Einsatzzweck dienen. Insbesondere sollten für die Verschlüsselung immer andere Schlüssel als für die Signaturbildung benutzt werden. Dies ist sinnvoll,

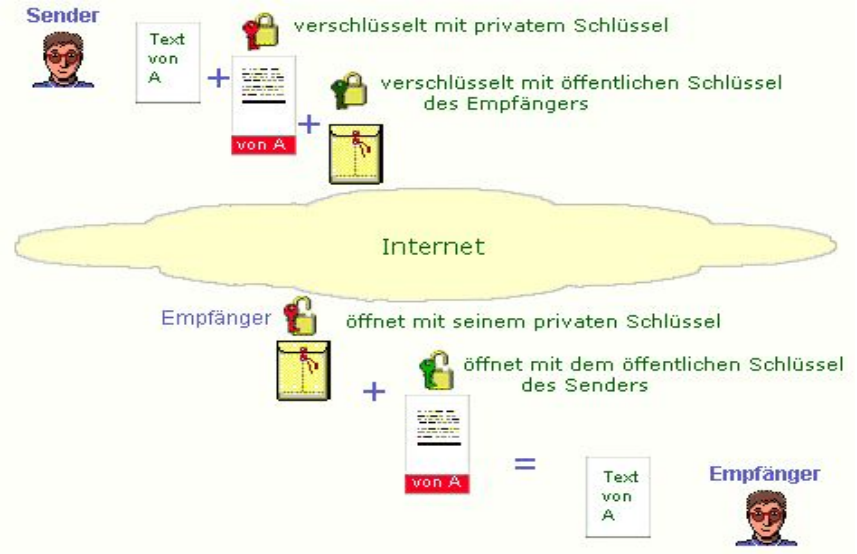
- damit bei der Offenlegung eines Schlüssels nicht alle Verfahren betroffen sind,
- da es manchmal erforderlich sein kann, Verschlüsselungsschlüssel weiterzugeben (Vertretungsfall),
- da es unterschiedliche Zyklen für den Schlüsselwechsel geben kann.

Schlüsselverteilung / Schlüsselaustausch

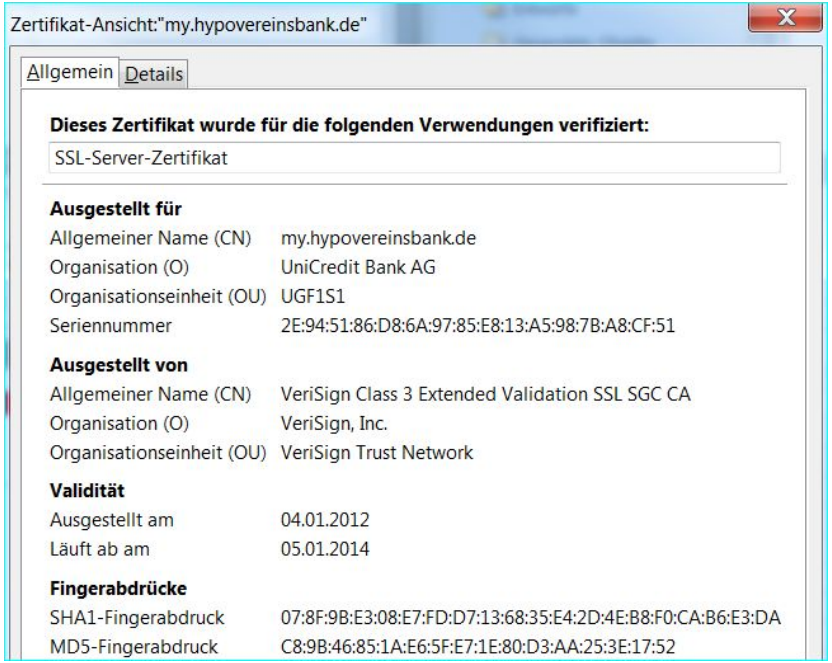
Verschlüsselung



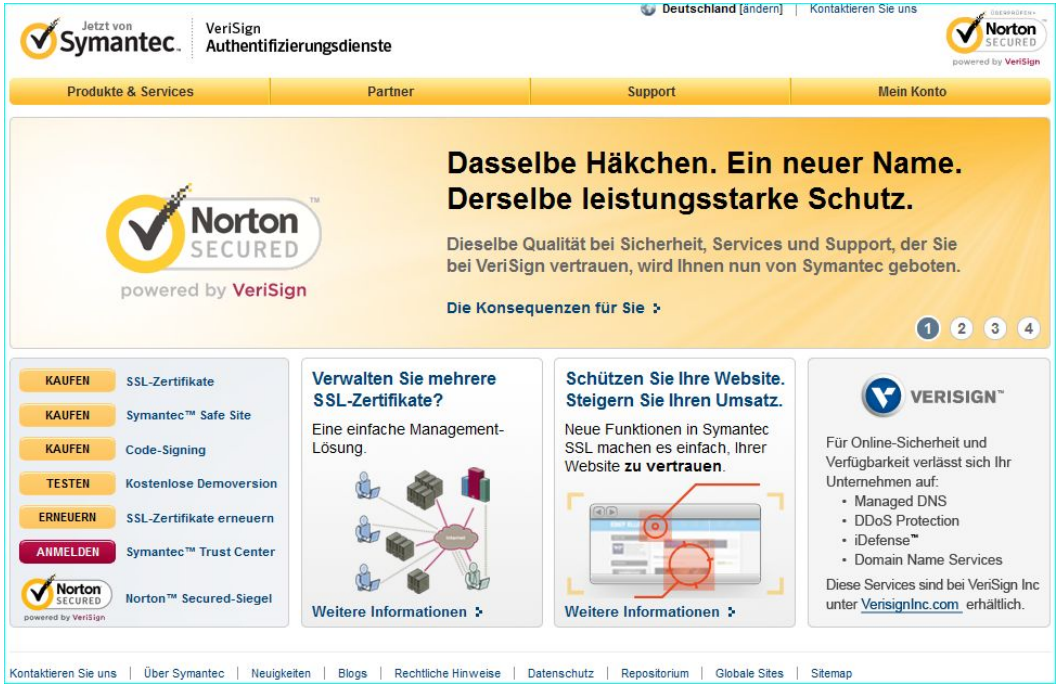




Digitale Zertifikate

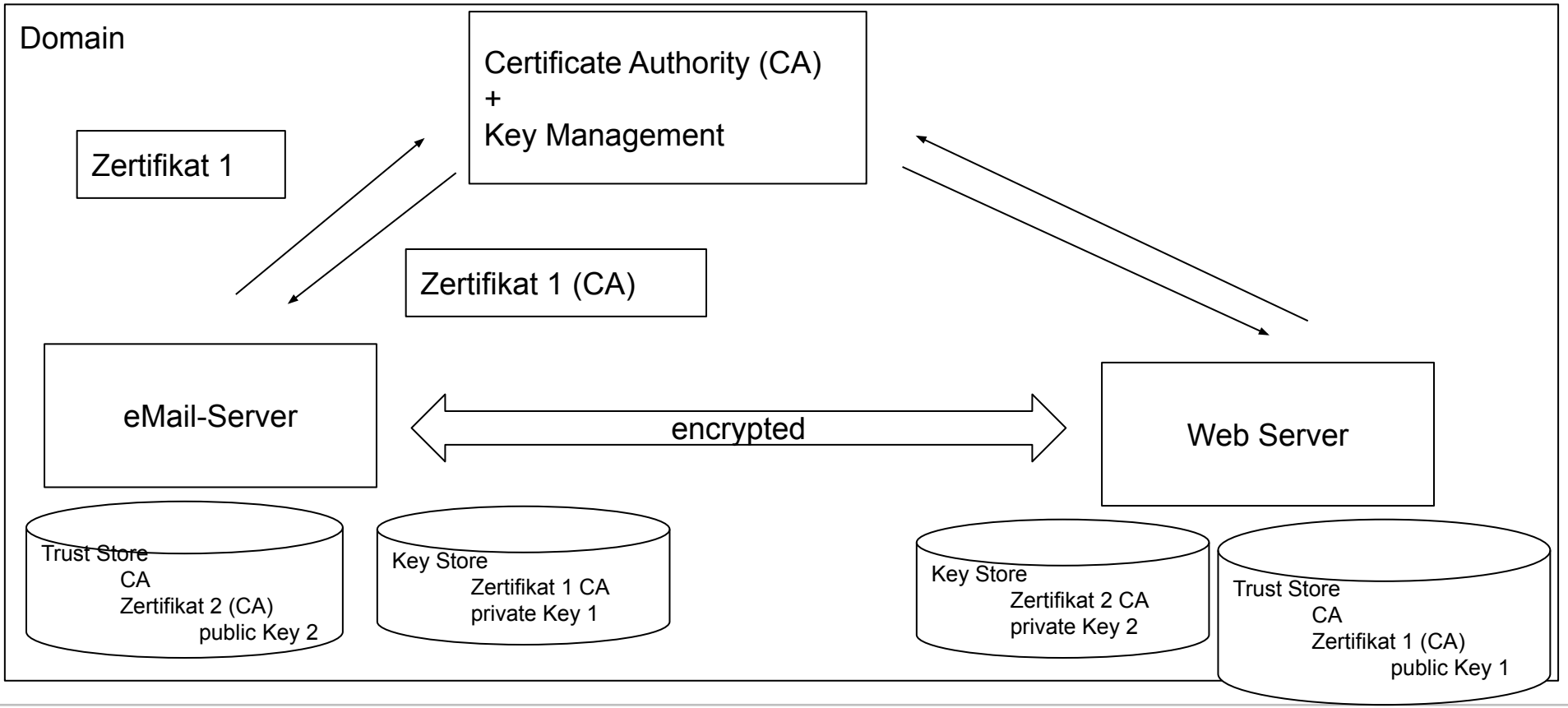


Zertifizierungs-Autoritäten



Review: Systeme innerhalb einer Domäne

integrata cegos



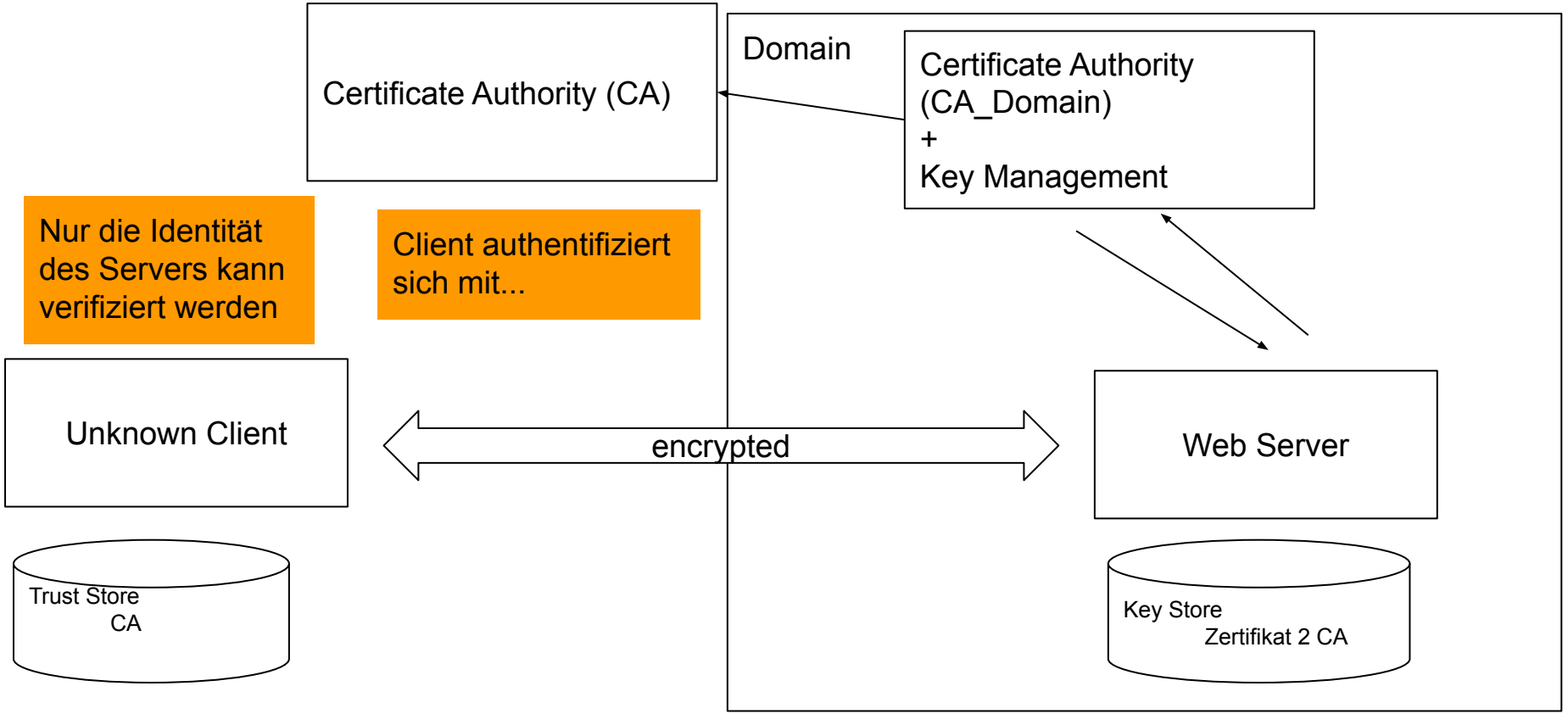
Review von SSL: Aufbau der verschlüsselten Verbindung

integrata cegos

- Asymmetrische Verschlüsselung wird ausschließlich dafür genutzt, einen zufällig erzeugten symmetrischen Schlüssel zu verteilen

Internet: Abgespecktes Identity Management

integrata cegos



Zertifikats-Fehler

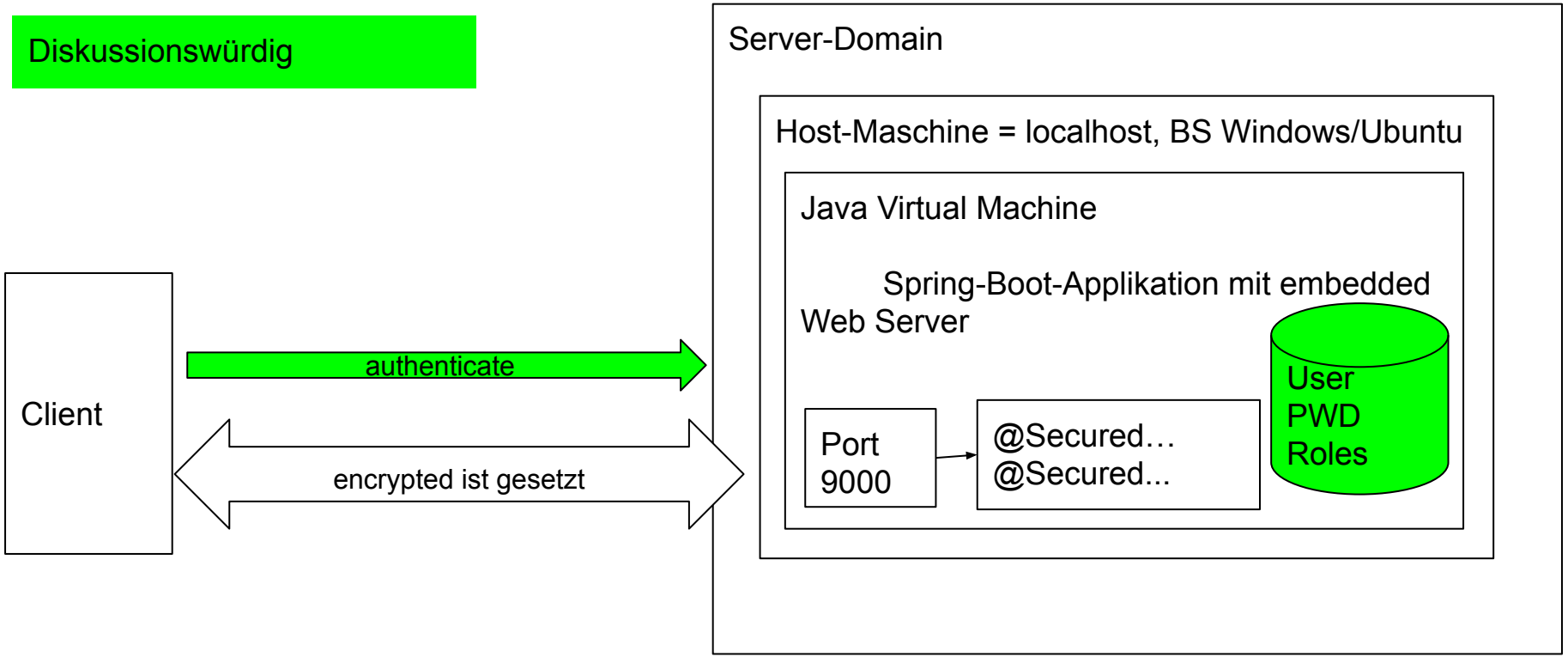
- Self-signed Zertifikate
  - Selbst-erstelltes Zertifikat, das von keiner CA gegengezeichnet wurde
    - Werden nicht automatisch akzeptiert
    - Im Internet: “Die Identität des Servers konnte nicht verifiziert werden”
      - Browser-Nutzer kann das Zertifikat trotzdem akzeptieren
    - Es gibt definitiv Produkte, die self signed nicht akzeptieren
  - Bedeutung für einfache Test-Server
- Zertifikat ausgelaufen
- Zertifikat kann nicht für diesen Host benutzt werden

Authentifizierung und Autorisierung

Client-Server-Anwendung

integrata cegos

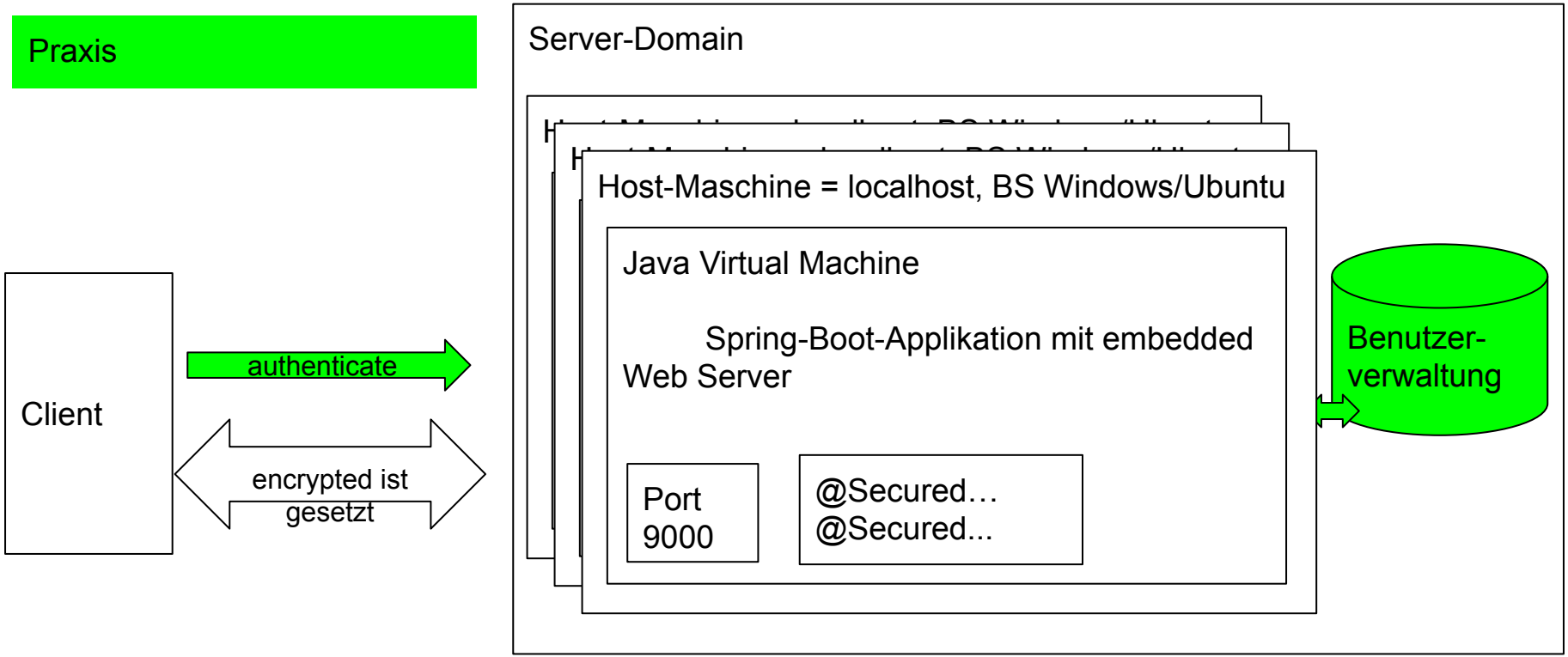
Diskussionswürdig



Client-Server-Anwendung

integrata cegos

Praxis

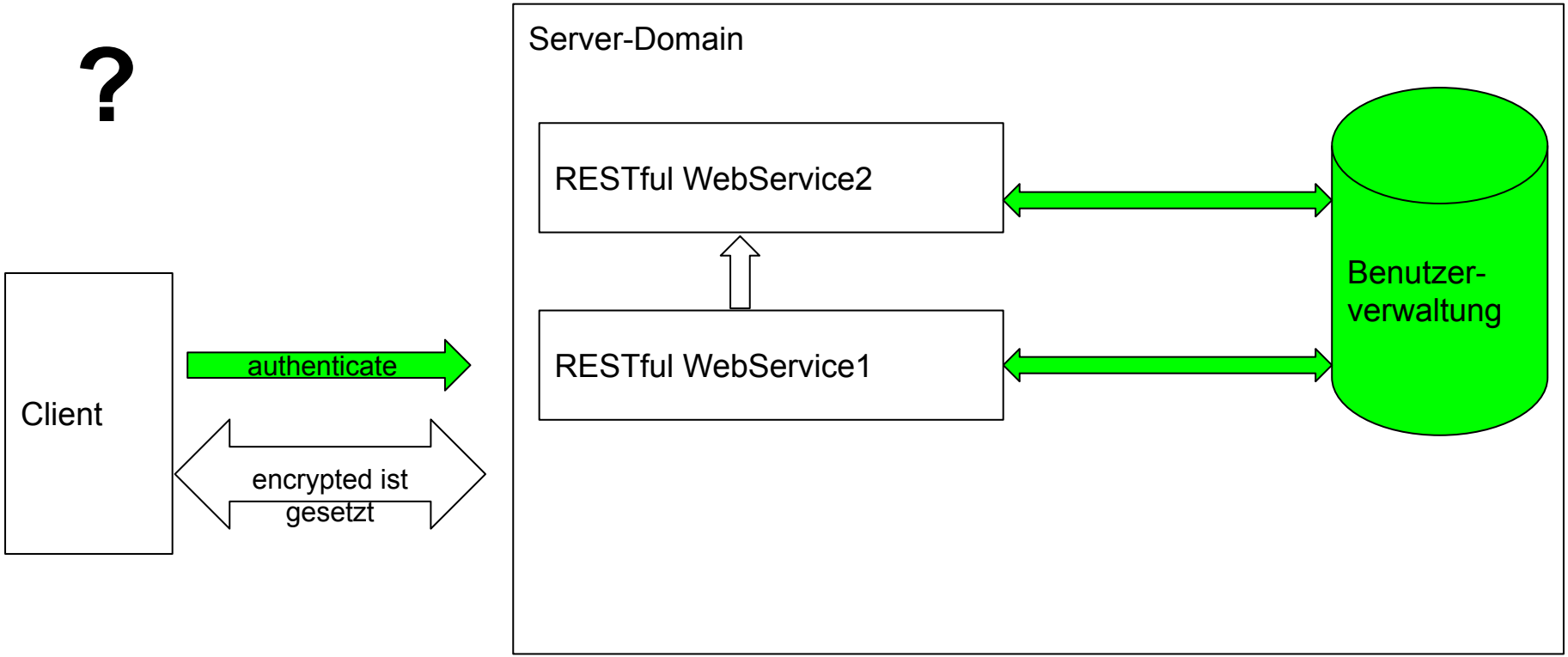


Anbindung an eine Benutzerverwaltung

integrata cegos

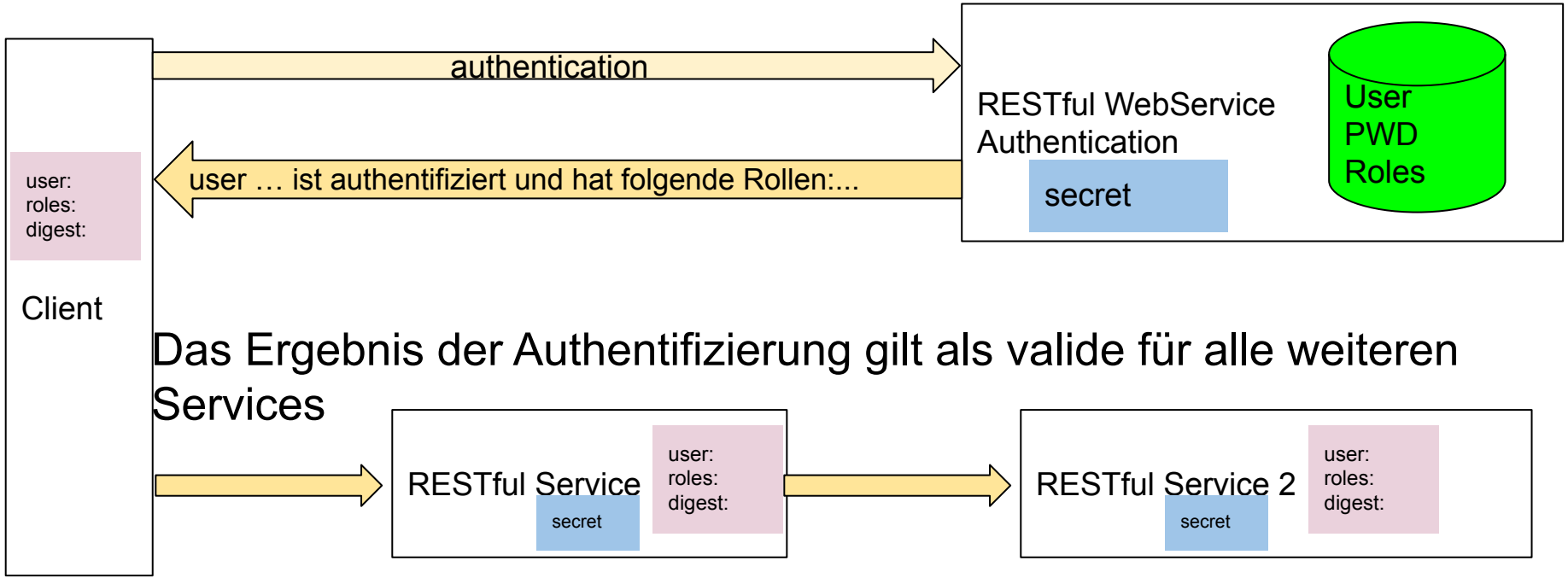
- Spring Security stellt für die typischen Benutzerverwaltungssysteme fertige Lösungen zur Verfügung
  - Datenbank mit User/PWD-Tabelle und User/Roles
  - Directory-Server, z.B. LDAP
- Heutzutage etwas “klassisch”

Service-orientierte-Anwendung



Ideen-Sammlung

- Authentifizierung wird delegiert an einen zentralen Service



Umsetzung: Spezifikation eines geeigneten Datenformats

integrata cegos

- Header, Payload, Signature
- Geeignetes Format
  - ~~Plain Text~~
  - XML
  - YAML
  - **JSON**
    - JSON Web Token, JWT, gesprochen “tschott”

Umsetzung

- https://github.com/Javacream/org.javacream.training.spring.security/tree/509217fe794d614f3dcccc365b76bf4abf6009fc

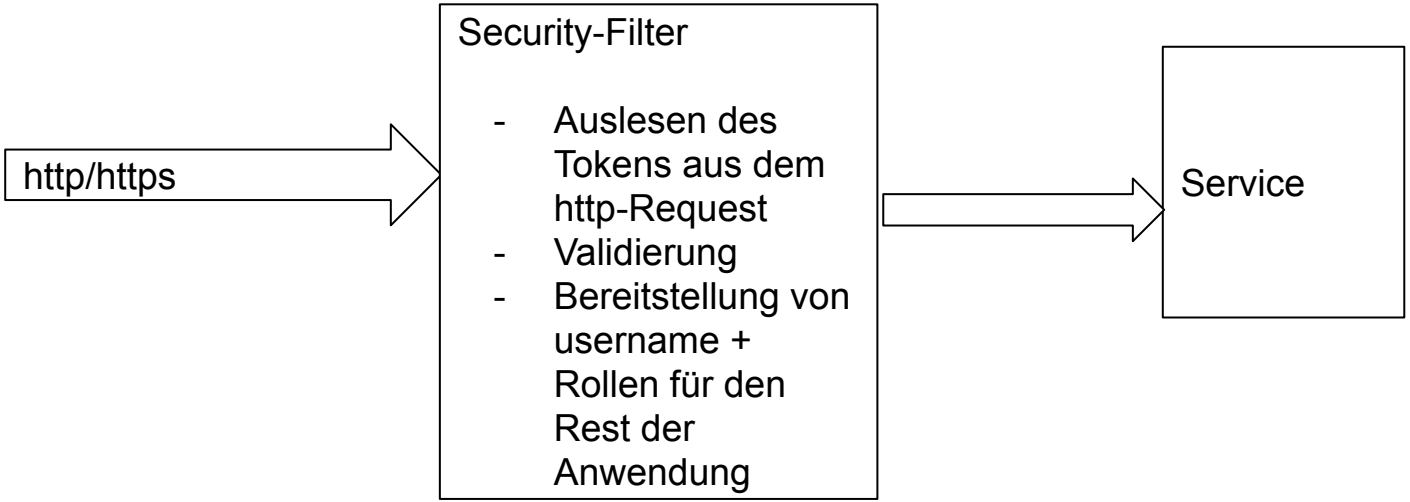
Sinnvolle Erweiterungen

- secret in der aktuellen Version muss den Services bekannt gemacht werden
  - Besser: Im Token-Service private Key, Verteilung des public Keys an die einzelnen Services
- Natürlich kann die Payload eines JWT separat verschlüsselt werden
  - insbesondere wenn keine verschlüsselte Verbindung vorausgesetzt werden kann
- JWT Encryption
  - Alternative zu den signierten JWTs

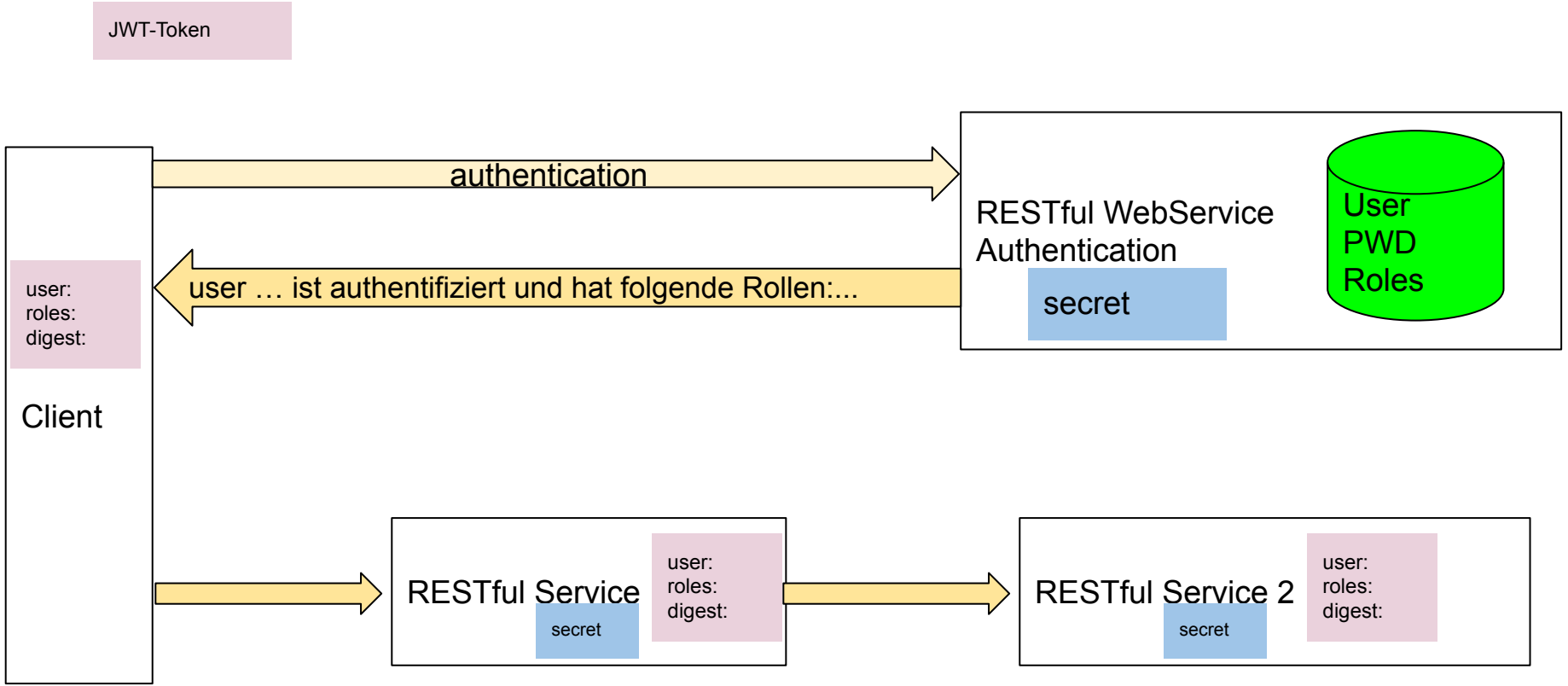
Exkurs: XML-Security für SOAP-WebServices

integrata cegos

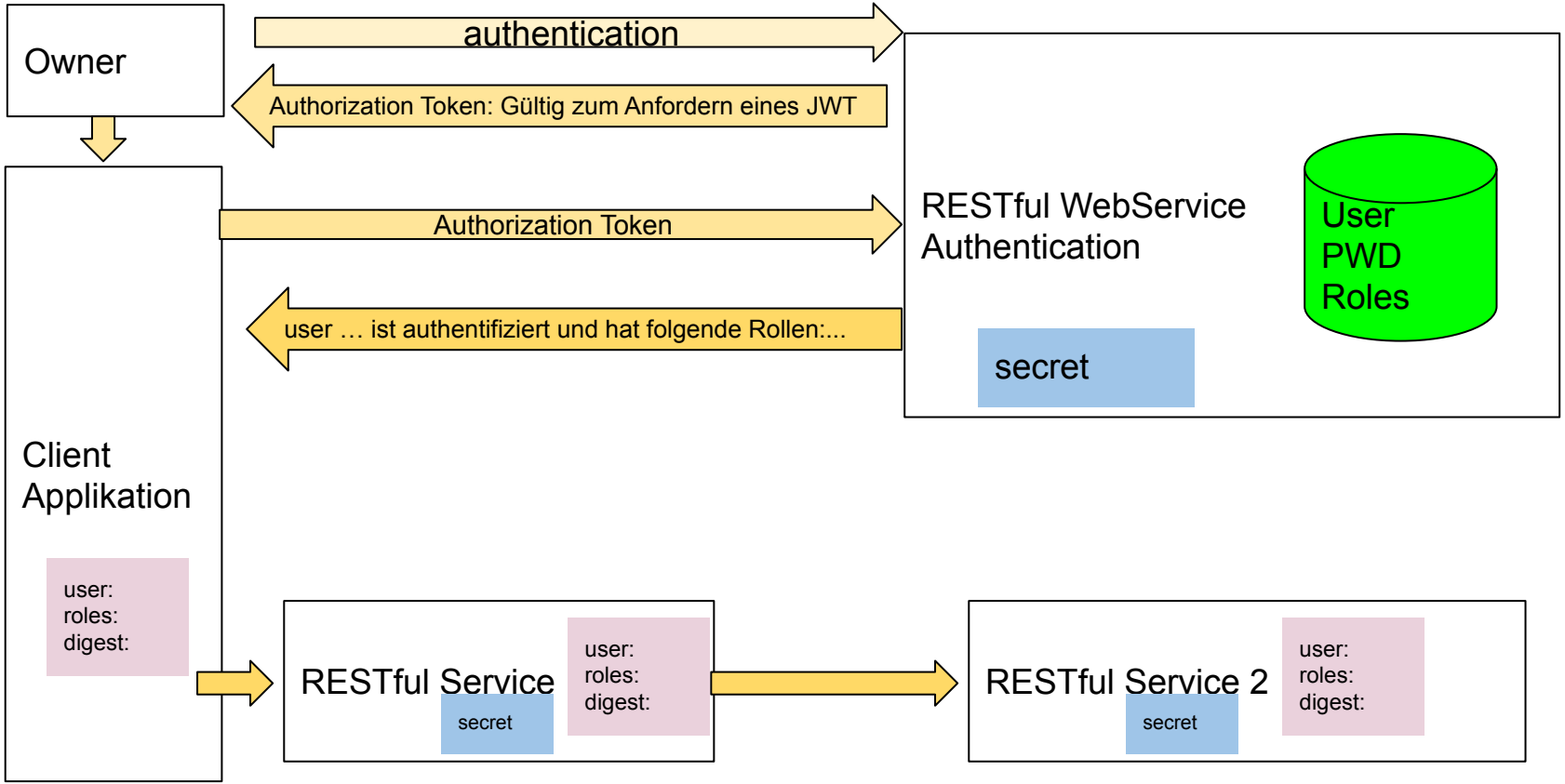
- Security Assertion Markup Language
  - Analog zu JWT eine Sprache zu Erstellung eines XML-basierten Tokens
- Damit kann jeglicher Endpoint mit einem Token aufgerufen werden



Aktueller Stand



Der Ressource Owner



Einführung in OAuth2

Endpoints des OAuth2 Providers

- 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”
  - Client-Applikation muss sich mit Client-Credentials authentifizieren
    - ClientID + “Client Secret”
    - Mit Hilfe des Authorization Codes wird das Access Token generiert

Optionale Endpoints

- Token Validierung
  - Access Token + Angeforderten Rollen
- Self Registration
  - Automatisches Erstellen des Clients ohne administrative Aktionen
- Liste
  - http://h2908727.stratoserver.net:9090/auth/realms/javacream/.well-known/openid-configuration

Der dritte Endpoint

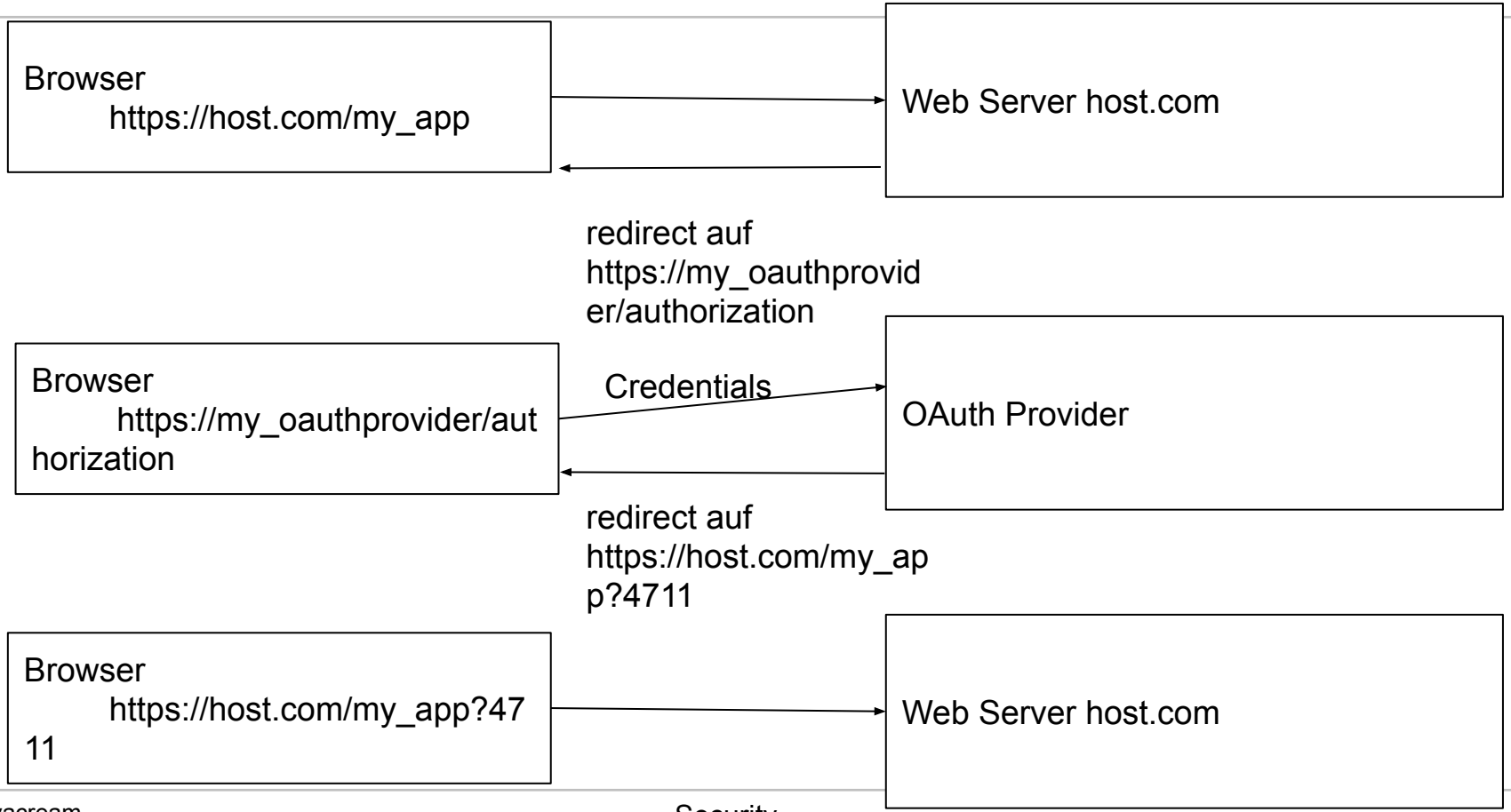
- Redirect Endpoint
- Bestandteil des Clients (!)
- Aufgabe:
  - Anmeldung des Resource Owners soll außerhalb der Client-App erfolgen
    - Client-App bekommt niemals Zugriff auf die Credentials des Resource Owners!

Authentifizierung mit Hilfe des Redirect Endpoints

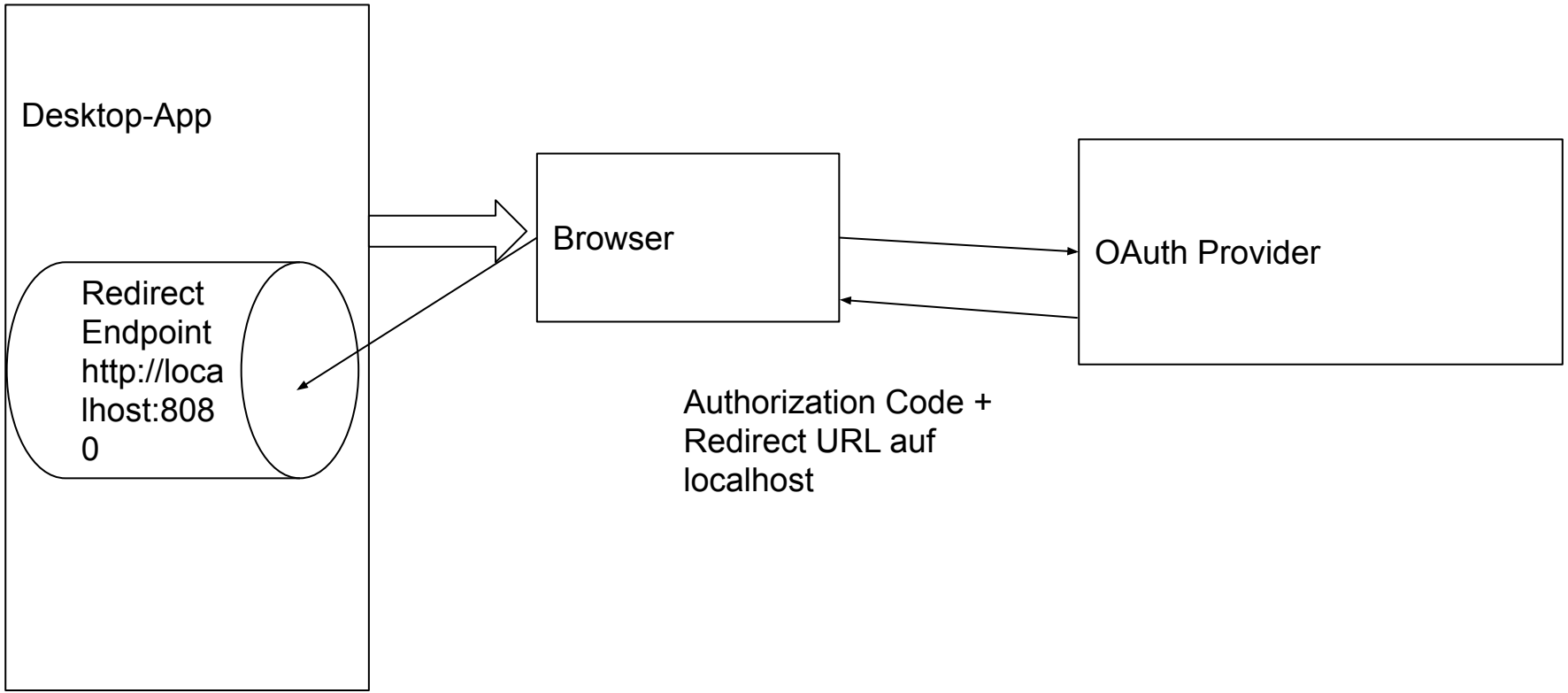
integrata cegos

- 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

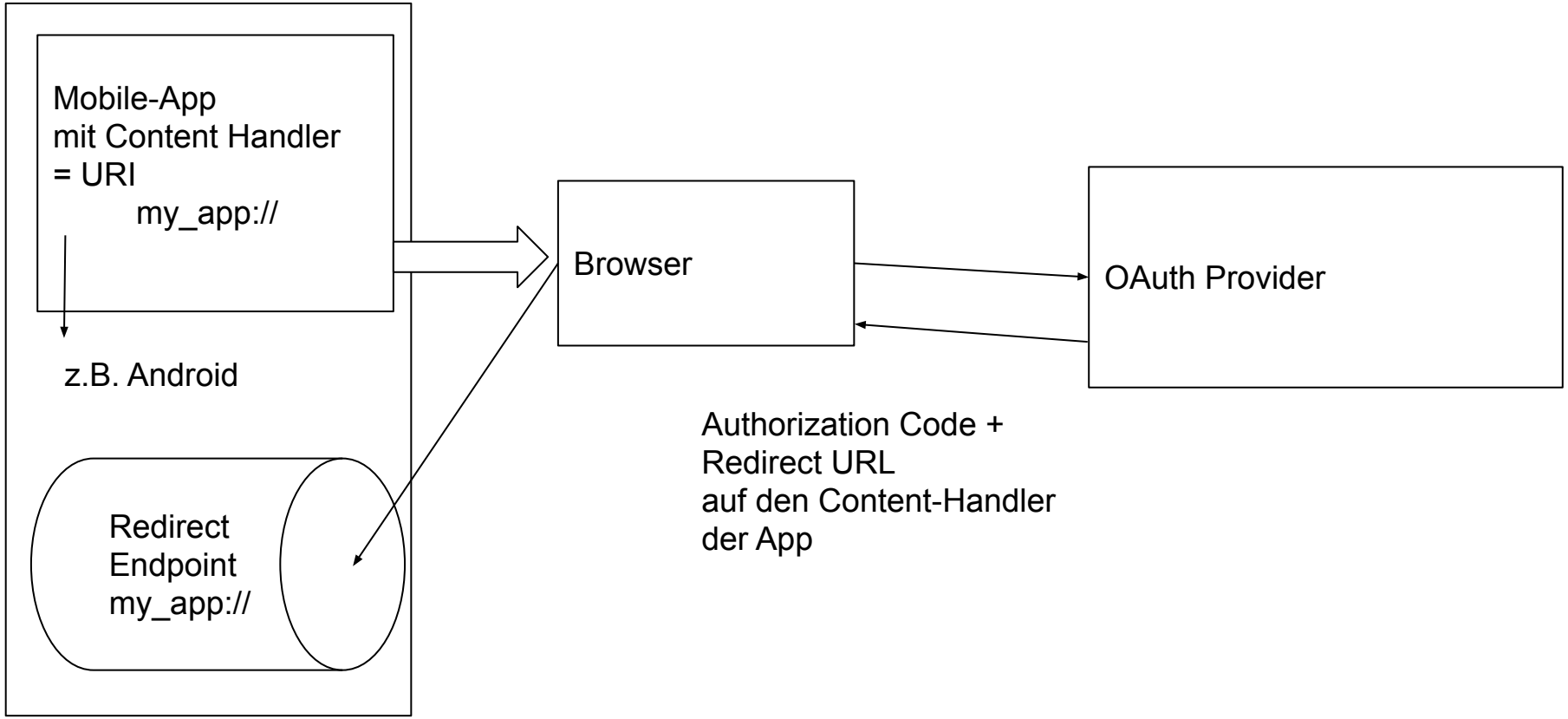
Redirect Endpoint im Browser



Redirect Endpoint in einer Desktop-Applikation



Redirect Endpoint Mobile App



OAuth2 Flows

Authorization Code Flow (3-leg Authentication)

integrata cegos

- 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

Lebensdauer der Tokens

- 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

Implicit Flow

- 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

Resource Owner Password Credential Flow

integrata cegos

- “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

Client Credential Flow

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

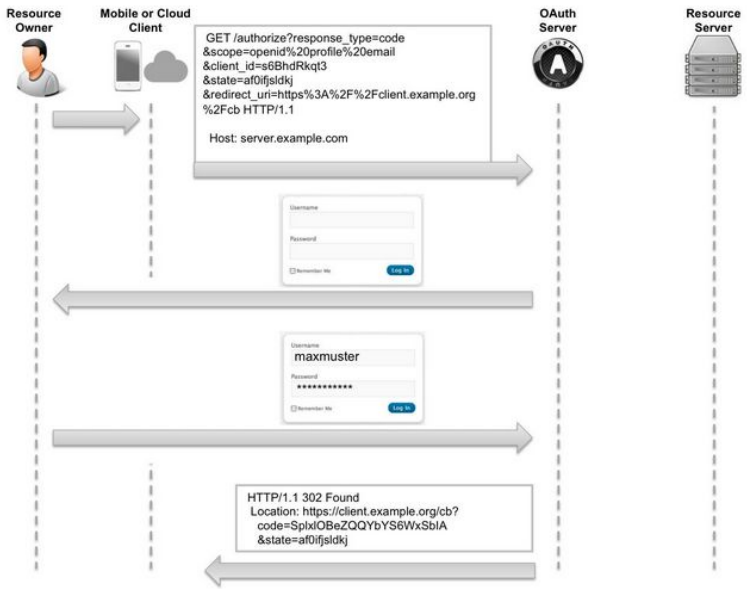
ToDo

- 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?

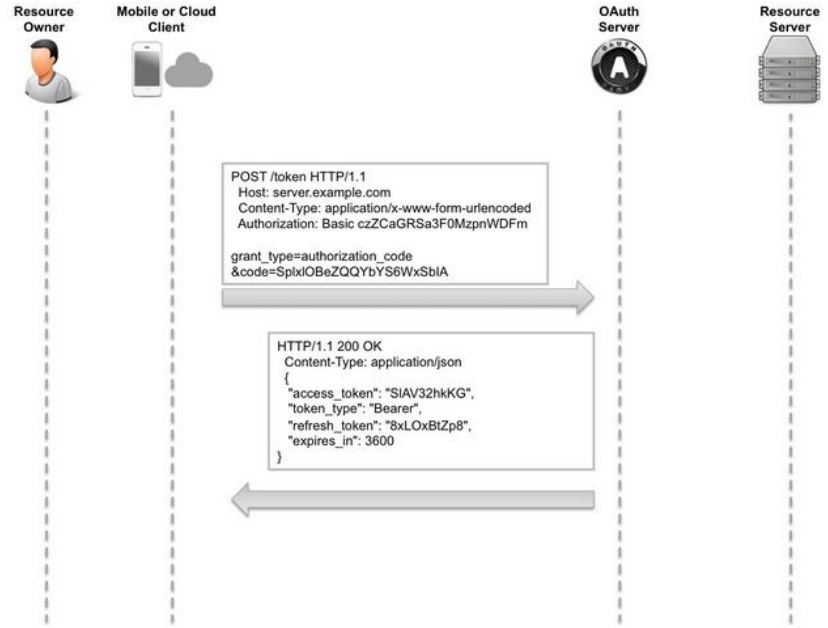
Authorization Flow

integrata cegos

Authorization Endpoint

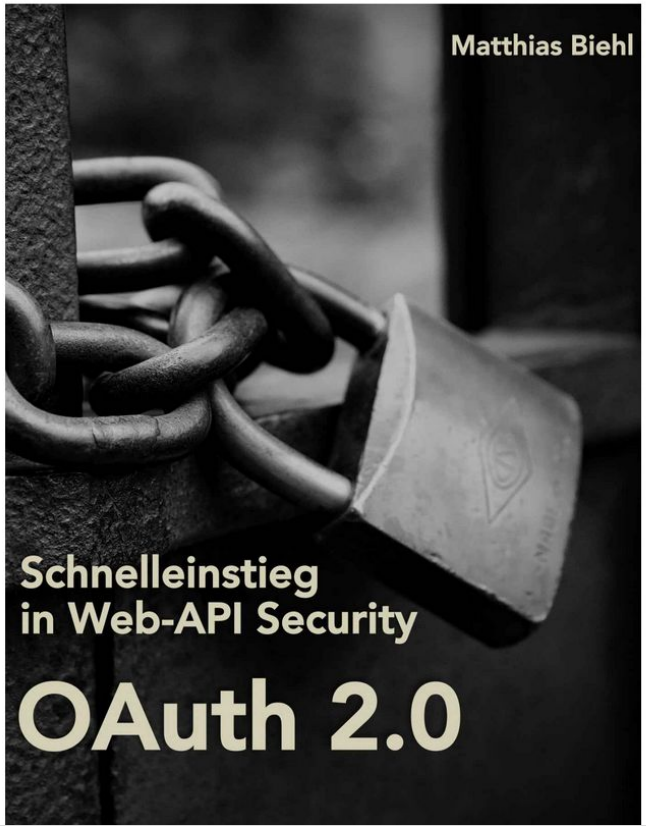


TokenEndpoint



Literatur zum Thema

integrata cegos



OAuth 2.0

Schnelleinstieg in Web-API Security

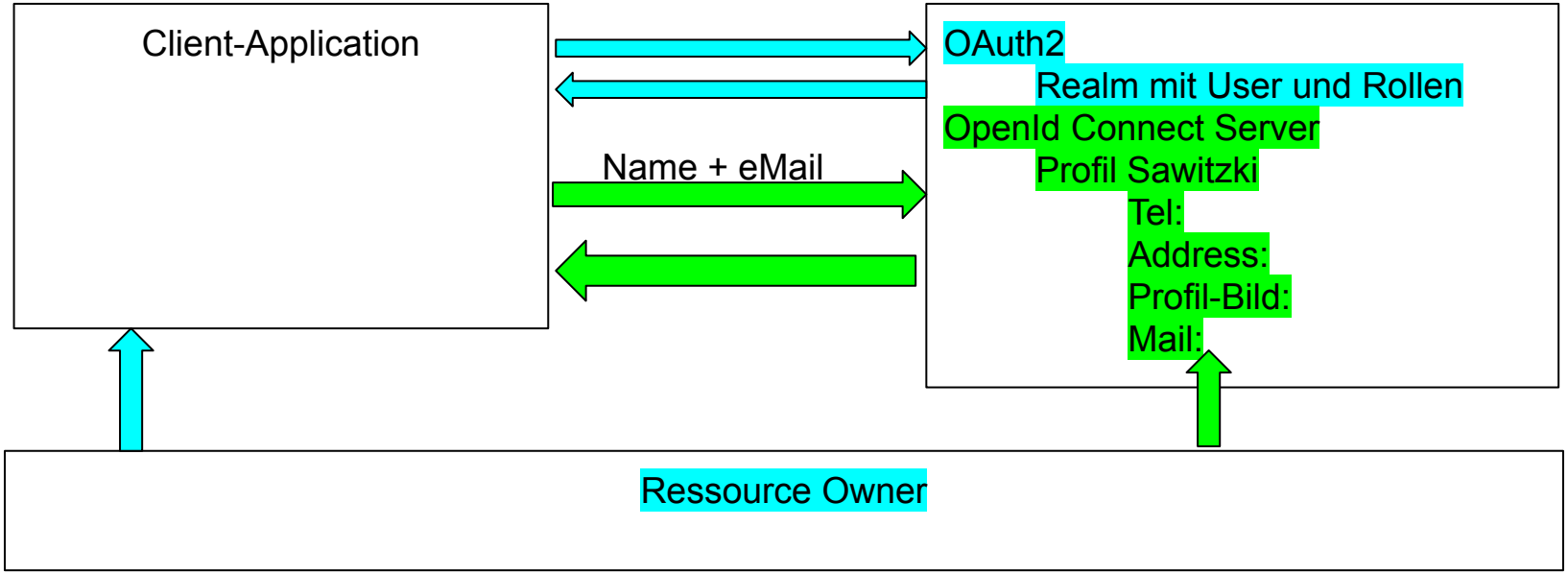
API University Series

www.api-university.com

OAuth2 und OpenId Connect

integrata cegos

- Problem: “Ich habe als Nutzer von Anwendungen keine Lust, meine Profil-Informationen immer wieder neu einzugeben”



Secure Java Programming

Der Security Manager

integrata cegos

- Ein integriertes Feature der Java-Klassenbibliothek
- Standardmäßig im Aufruf der Java Virtual Machine nicht aktiv
  - Dies verletzt die Regel “Secure by Default”
  - Aktivierung -Dsecurity.manager
- Der SecurityManager prüft jeglichen relevanten Ressourcen-Zugriff und kontrolliert den Zugriff
  - Standard-Verhalten: Zugriffe sind verboten
- Über Policies können Permissions vergeben werden
  - z.B. FilePermission: Dateien in einem bestimmten Verzeichnis dürfen verändert werden

Ergänzungen

- Es ist unmöglich, korrumpierten ByteCode in die Java Virtual Machine zu laden
  - Das ist Aufgabe des ByteCode-Verifier
    - Ausnahme: Java Card
- ? -> später

Best Practice

- Java-Prozesse sollen nur mit aktiviertem SecurityManager ausgeführt werden
- Vorsicht:
  - In der Praxis führt das zu beträchtlichem Aufwand
    - AllPermission ist keine Lösung
  - Besser
    - Laufenlassen der Anwendung mit aktiviertem security-Debuging

Reale Praxis

- Zugriffsbeschränkungen auf Dateisystem sowie das Netzwerk werden über das zugrunde liegende Betriebssystem realisert
  - Java-User mit “least priviliges”
  - Firewall
- Der Java-Prozess läuft in einem Container (-> später)

Guidelines für sichere Java-Programmierung

integrata cegos

- https://www.oracle.com/java/technologies/javase/seccodeguide.html
  - Grundlage für Code-Scanner
    - Sonar, XRadar, …
- These: “Java Programmiersprache ist per se sicher”
  - Keine Memory Leaks
  - Keine unzulässigen Speicher-Manipulationen
  - Stimmt nicht!
    - Thread t = new Thread(new Runnable(){... //Variablen etc});
    - Ressourcen-Zugriffe über Java-Klassen werden im Endeffekt native ausgeführt
    - Java Native Interface als Schnittstelle zur C-Programmierung
    - Native Memory, Off-Heap-Memory
      - https://java.integrata-cegos.de/off-heap-memory-in-java/

Regel: Verwenden Sie etablierte Komponenten und Bibliotheken!

integrata cegos

- Sämtliche “komplizierten” und kritischen Code-Sequenzen haben in einem Anwendungsprogramm nichts verloren!
  - Multithreading
  - Client-Server-Programmierung
  - Erzeugung von Business-Objekten
  - Ressourcen-Zugriff auf Datenbanken, Messaging-Systeme, …
- Beispiele
  - Multithreaded-Server -> JEE ApplicationServer
  - RESTful WebServices z.B. mit Spring MVC oder Jersey (JAX-RS)
  - Context and Dependency Injection (z.B. Spring)
  - Datenbank-Zugriffe mit JPA und z.B. Hibernate

Sichere Umgang mit Komponenten und Bibliotheken

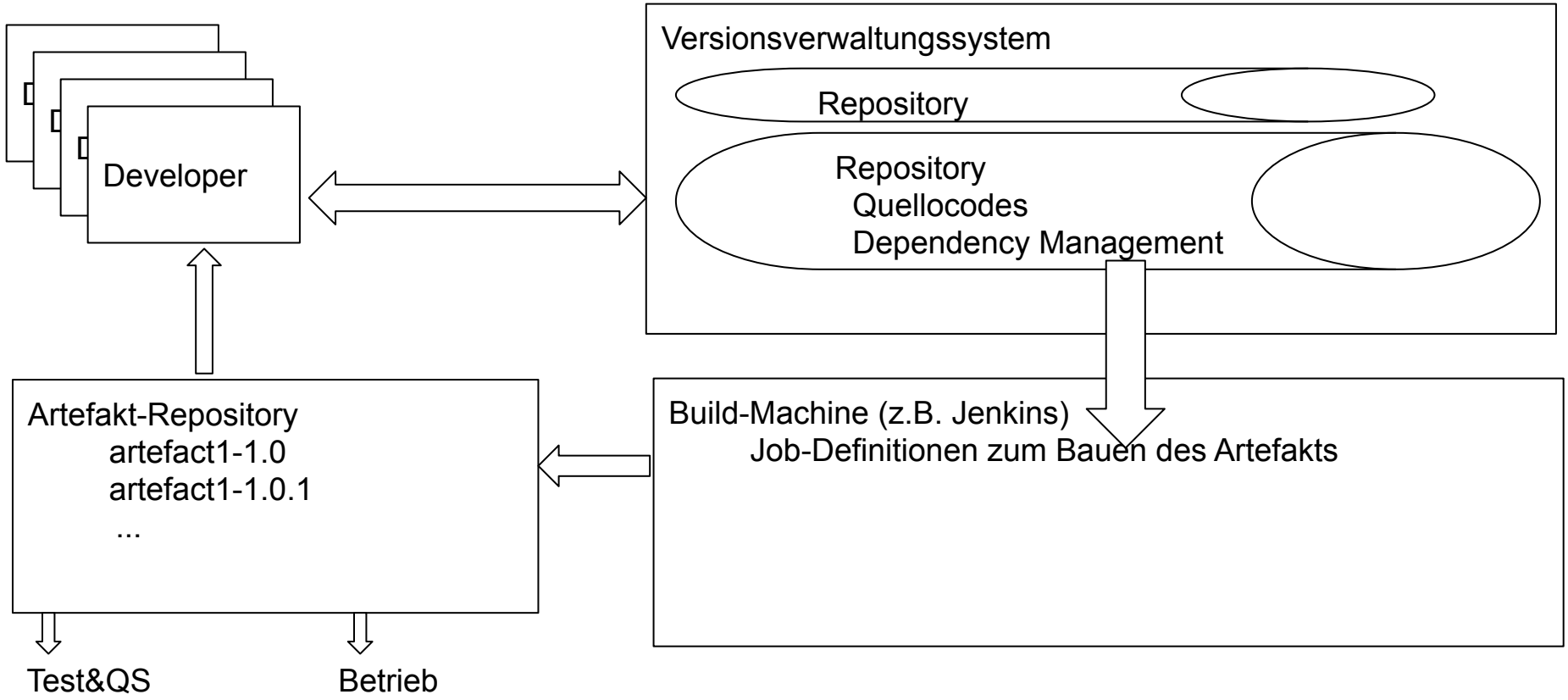
integrata cegos

- Eindeutige Identifikation einer Komponente muss gewährleistet sein!
  - Versionierungs-Konzept mit sauberen Versionierungs-Schema
    - major.minor.patch.qualifier
      - 1.0, 1.0.1, 1.1, 2.0
  - Runtime-Artefakte werden gemäß dieser Versionierung eindeutig, reproduzierbar und portabel aus Quellcode erstellt
    - in Java ist dies gewährleistet (!)
  - Identifikation ist nun möglich über eine Koordinate
    - groupId
      - Hierarchie
    - artifactId
      - Frei wählbarer Name
    - version
      - Versionsnummer

Dependeny Management

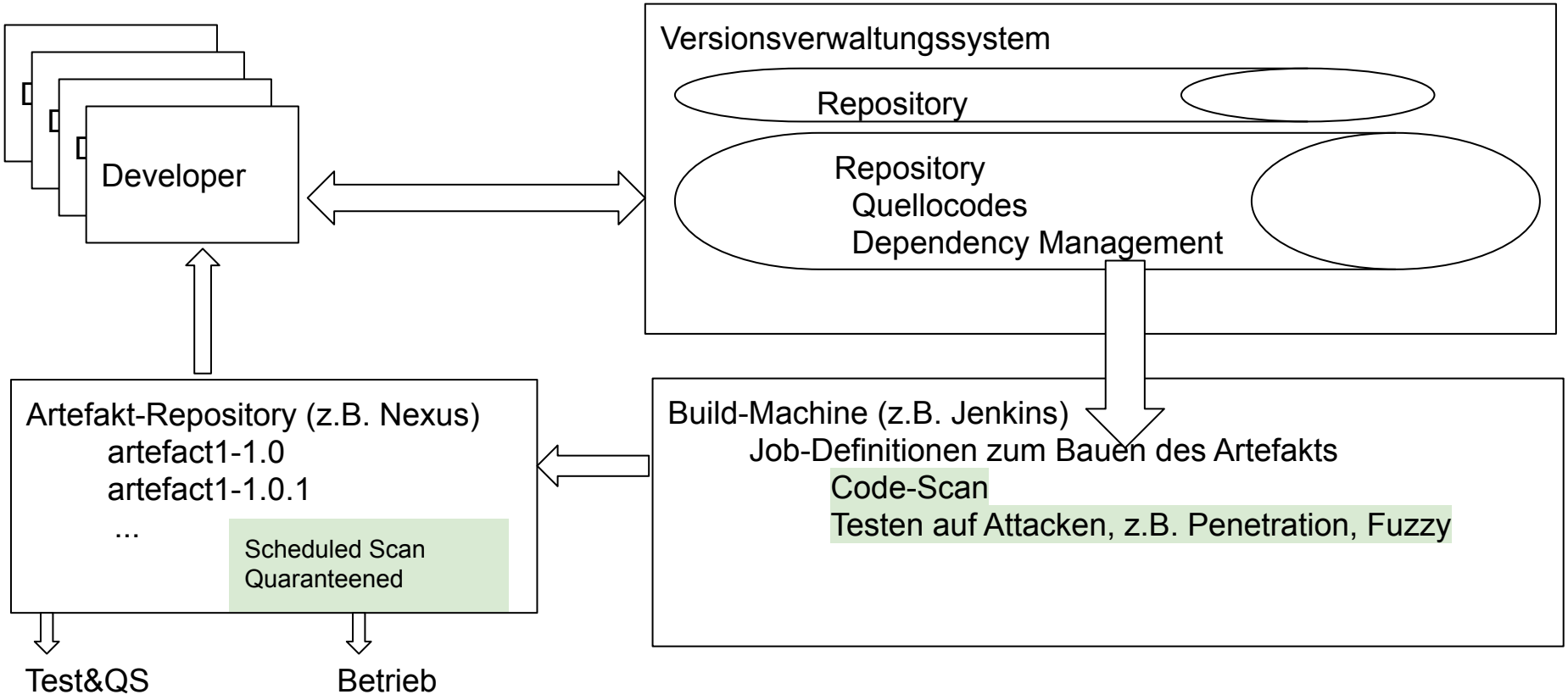
- Jede Anwendung deklariert eine Reihe von Koordinaten, die zum Bauen und Betrieb der Anwendung notwendig sind
  - z.B. pom.xml
- Damit wird für das eigene Projekt ebenfalls ein portabler und reproduzierbarer Build-Prozess realisiert
-

Bauen und Ablegen der Artefakte



Bauen und Ablegen der Artefakte: Security

integrata cegos



Hardening der Umgebung

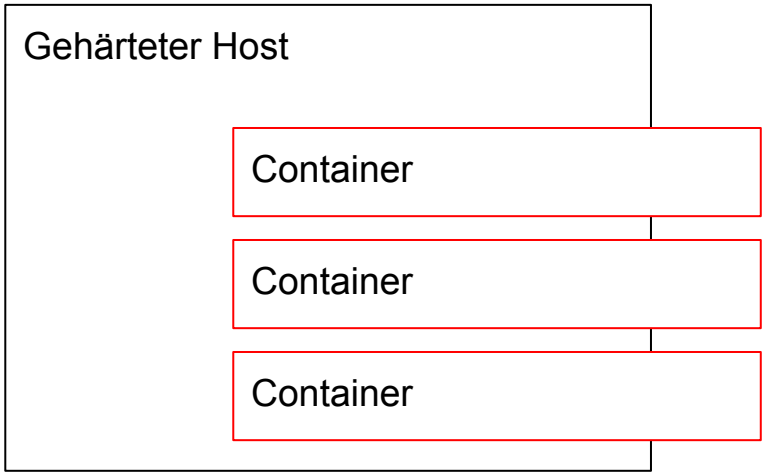
Physikalische Hardware

- Verschlüsselte Datenablage
  - Festplatte
  - SIM-Karte
- Verschlüsselung der Kommunikation
  - Spezielle Netzwerk-Karten
  - Eher unüblich, obwohl technisch möglich
- Sicheres RAM

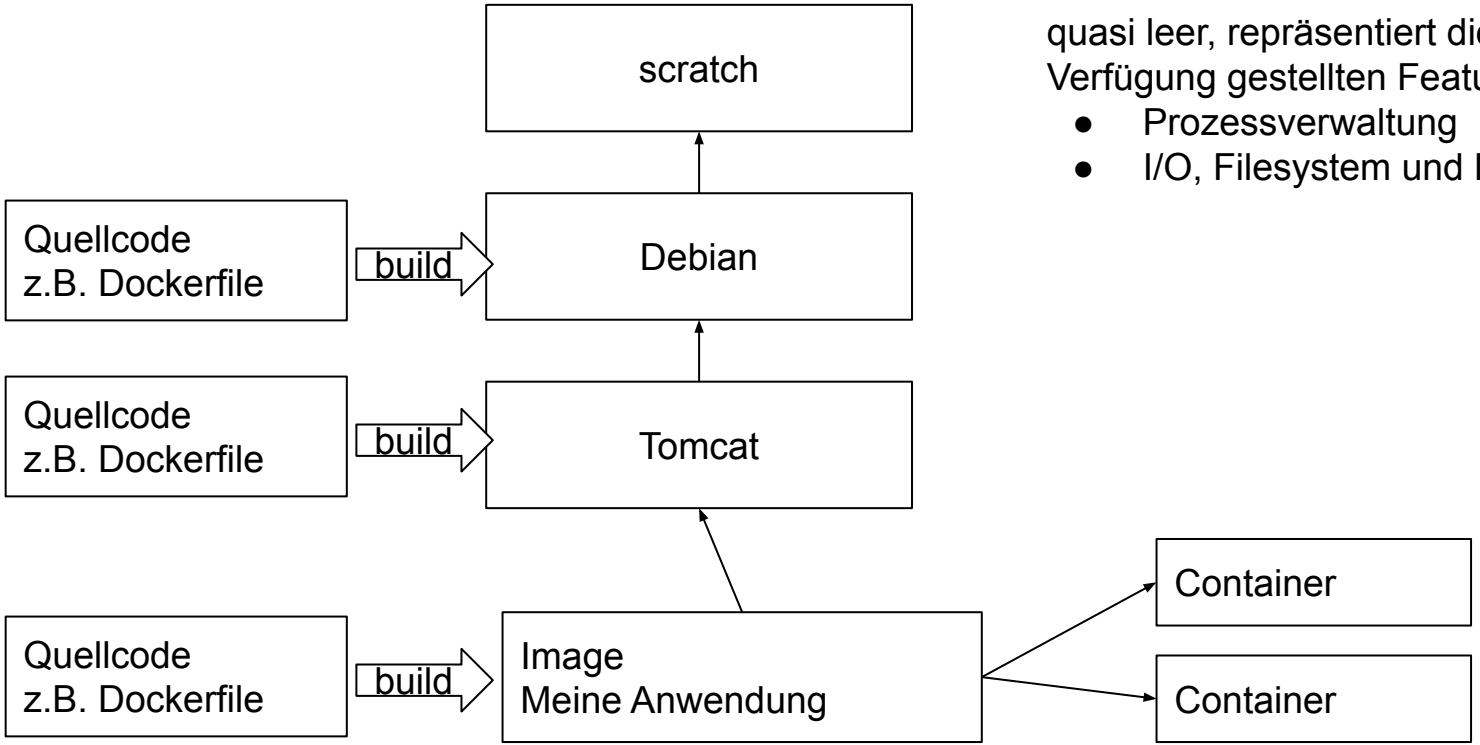
Betriebssystem-Ebene

- Kein Benutzer hat “wirklich” root-Rechte
  - das gilt auch für den Administrator
- Einspielen der Security-Patches
- Selbst-kompiliertes Kernel wird benutzt
- In der Summe aufwändig
  - Nicht Bestandteil einer “normalen” Installation einer Anwendung
  - Gehärtete Hosts dienen als Ausführungsschicht für
    - Anwendungs-Images (Virtual Box, VMWare)
    - **Container** (Docker)

Container und Host



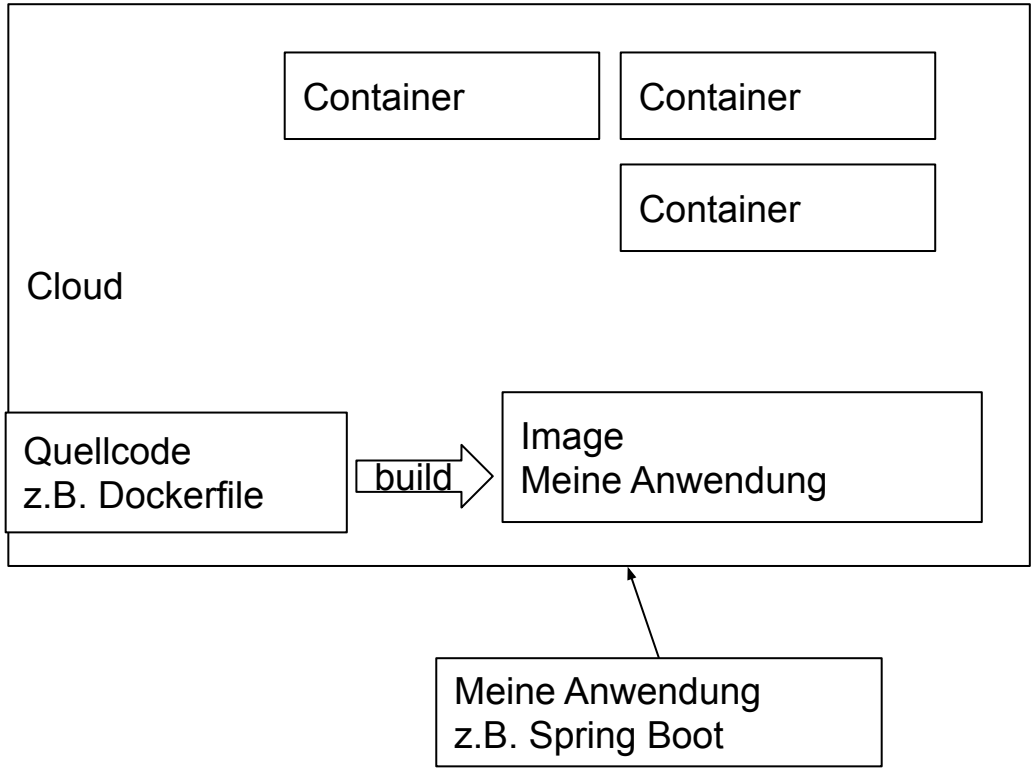
Container und Images



quasi leer, repräsentiert die vom Host zur Verfügung gestellten Features

- Prozessverwaltung
- I/O, Filesystem und Netzwerk

Container und Images



quasi leer, repräsentiert die vom Host zur Verfügung gestellten Features

- Prozessverwaltung
- I/O, Filesystem und Netzwerk

Härten eines JEE-Applikationsservers

integrata cegos

- Läuft auf einer gehärteten JVM
  - Security Manager ist aktiv
  - Permissions können auf Basis eines signierten Java-Archivs gegeben werden
- Entfernen aller vorhandenen Demo-Applikation
- Entfernen aller nicht-benötigten Services
  - z.B. http-Port, wenn nur über SSL kommuniziert werden darf
  - z.B. native JMX
- Admin-und Monitoring-Anwendungen immer auf einem anderen Netzwerk betreiben
- Review und Konfiguration der Log-Ausgaben und der Exception-Meldungen
  - Http-Header in Responses

Exkurs: Ist Docker sicher?

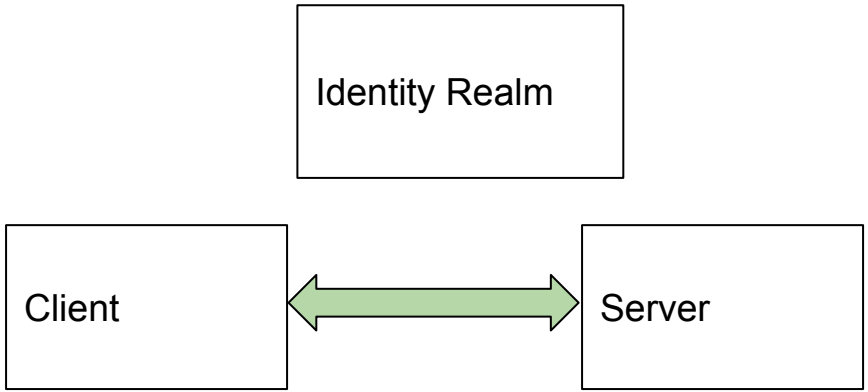
integrata cegos

- Internet-Recherche
  - “Docker verlangt die Ausführung als root”
  - Vorsicht
    - Nur die Docker-Engine verlangt root-Nutzer
    - Docker-Engine läuft eh auf einem gehärteten Host
- Docker-Image Build und Docker Engine sind völlig unabhängig voneinander
  - Ausführung von Containern kann auch mit anderen Werkzeugen passieren
    - LNX: Native Linux Container
    - Podman (Oracle)

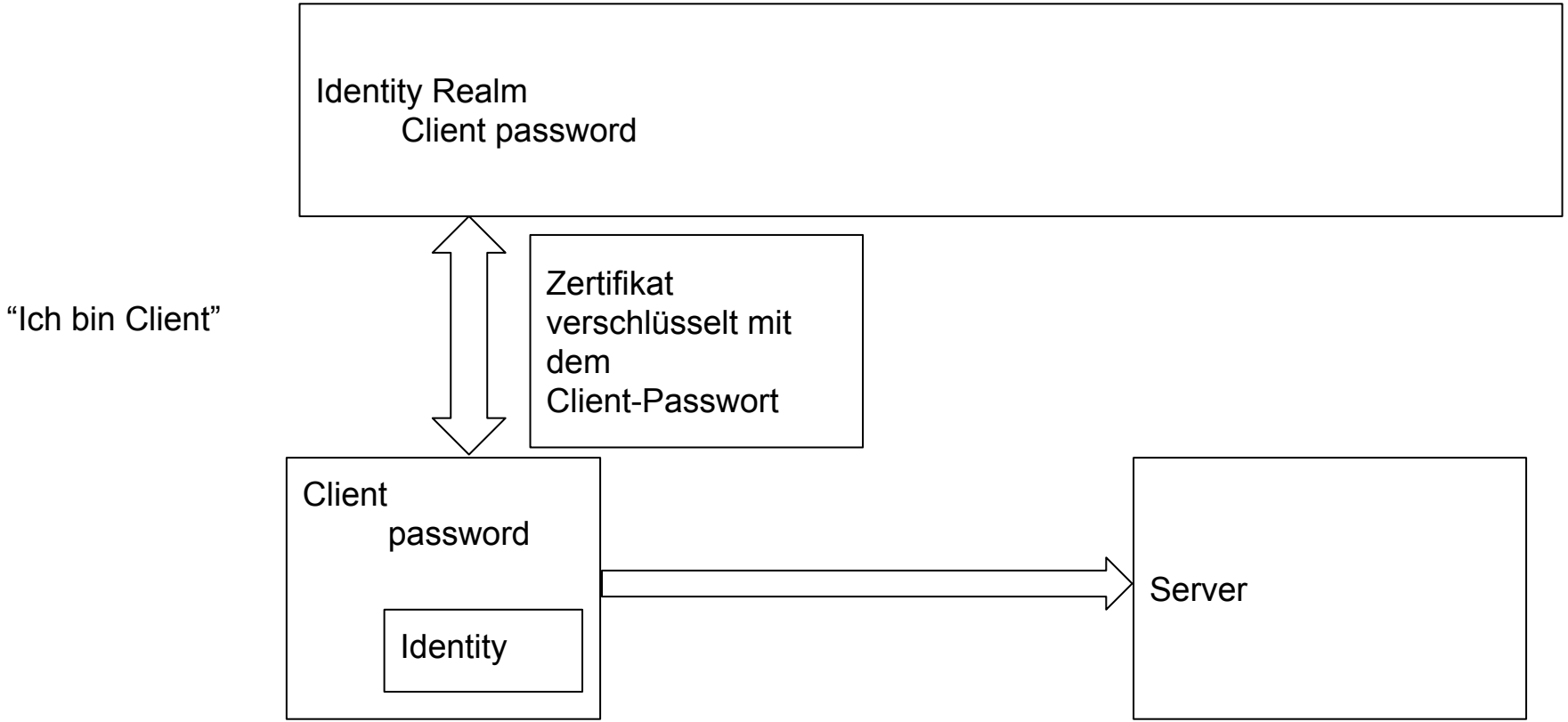
Kerberos

Anforderungen

- Kommunikation erfordert keine verschlüsselte Verbindung
  - Damit dürfen keinesfalls Credentials über das Kommunikationsprotokoll übertragen werden!
- Alle Daten werden immer in verschlüsselter Form übertragen
- Identities werden in einem Realm verwaltet

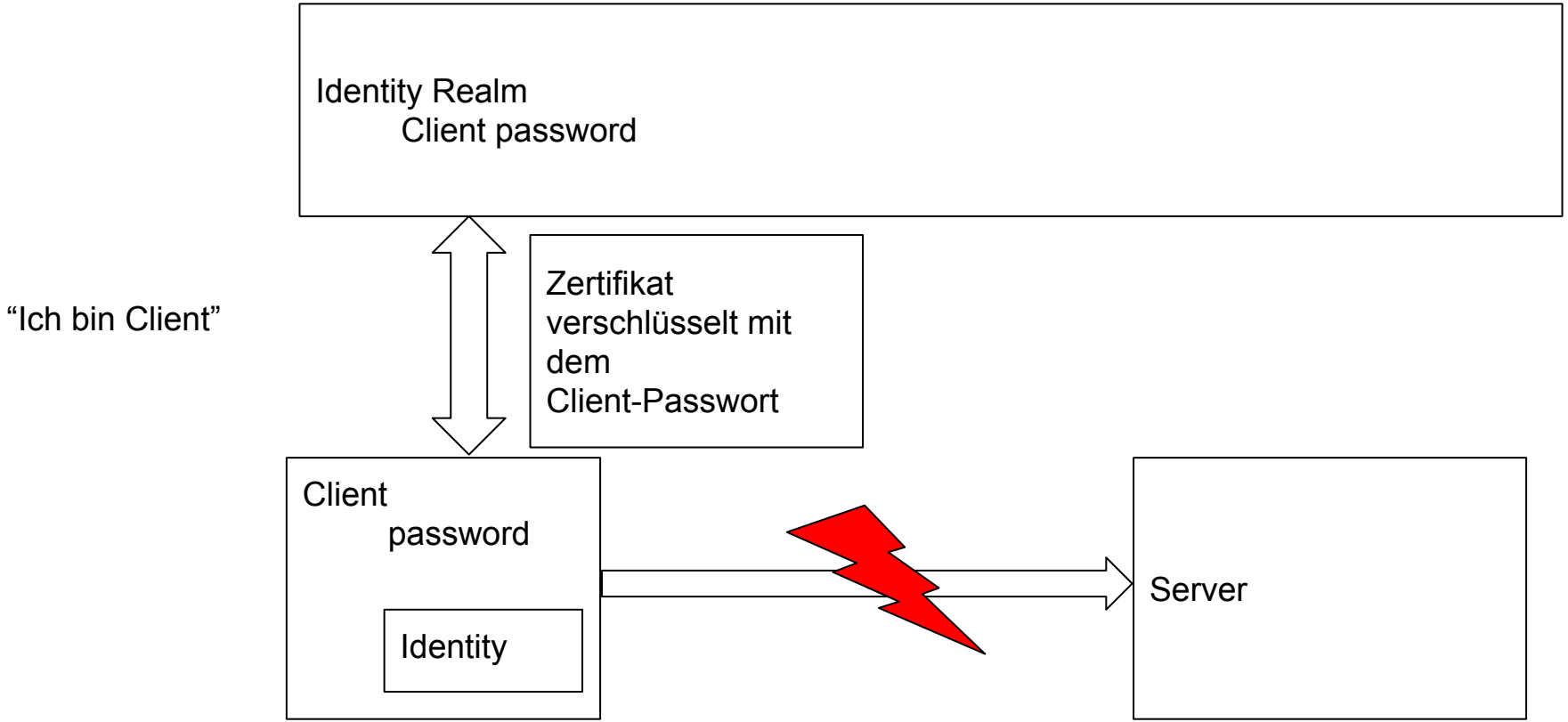


Konzeptuelle Umsetzung

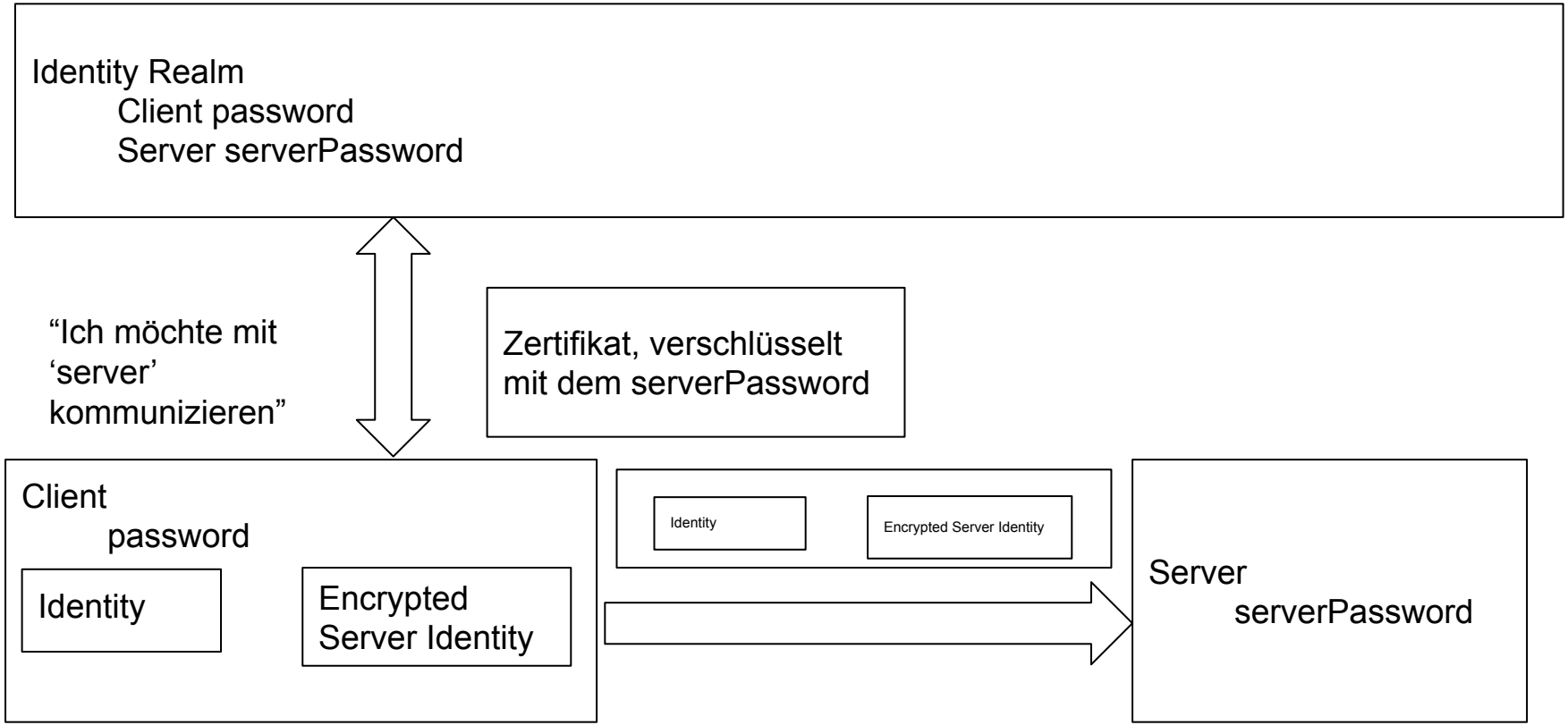


Umsetzung: Naiv: Damit wäre ich fertig

integrata cegos



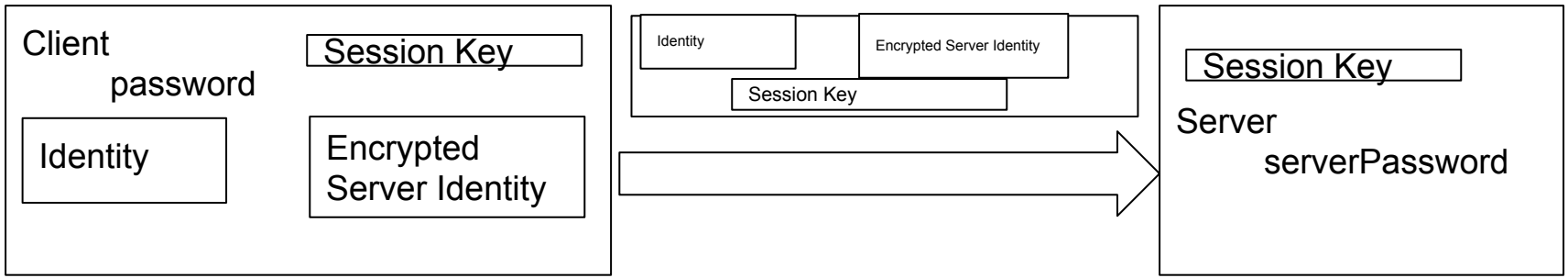
Konzeptuelle Umsetzung



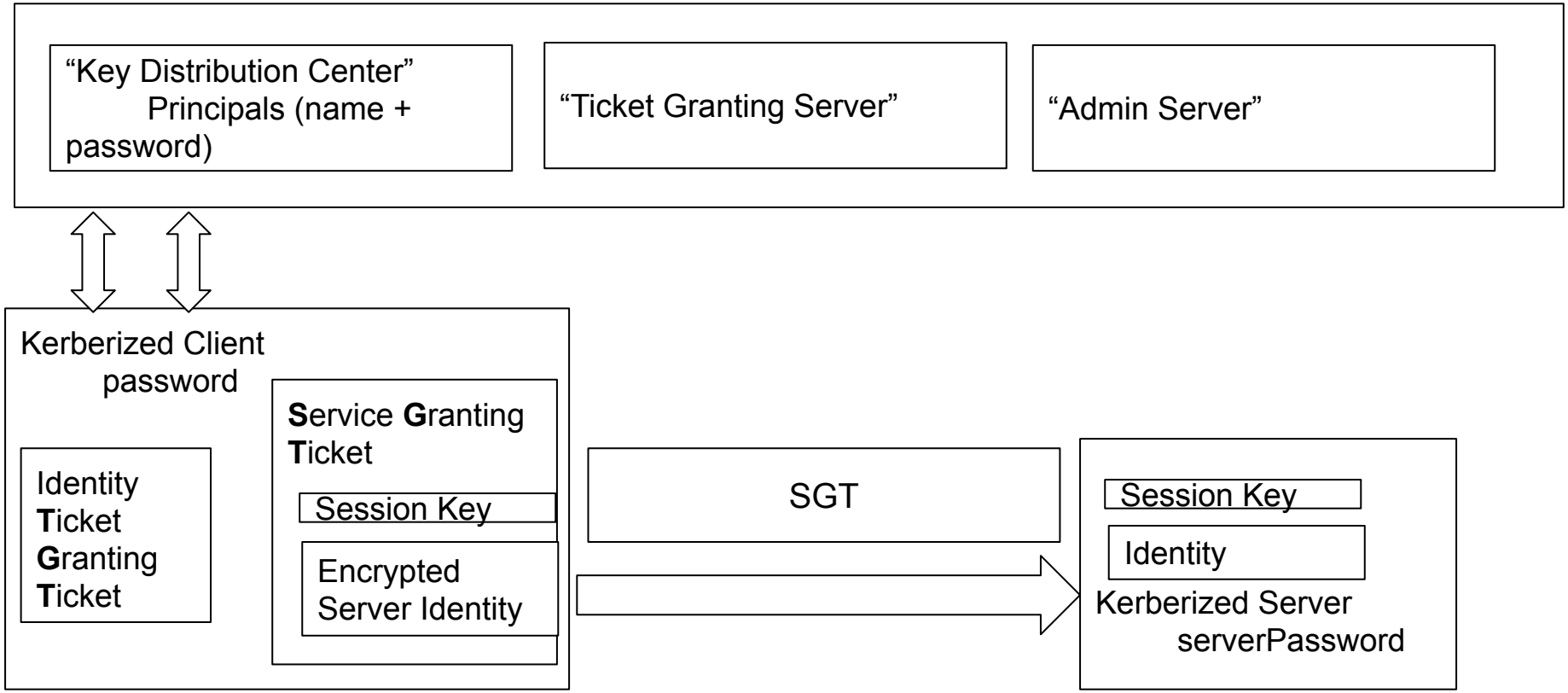
Konzeptuelle Umsetzung

integrata cegos

Identity Realm
Client password
Server serverPassword



Kerberos


“Key Distribution Center”
Principals (name + password)
“Ticket Granting Server”
“Admin Server”
Kerberized Client
password
Identity Ticket Granting Ticket
Service Granting Ticket
Session Key
Encrypted Server Identity
SGT
Session Key
Identity
Kerberized Server
serverPassword

Hinweise

integrata cegos

- 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