Hochschule **Rosenheim**
University of Applied Sciences

Verteilte Verarbeitung

Kapitel 14 –
REST Sicherheit

Sicherheit

Achtung:

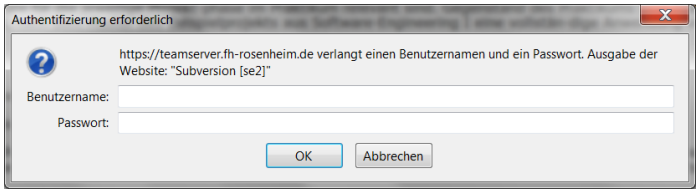
- Kryptologie ist Stoff von GDI 1 (Prof. Dr. Schmidt) MD5, SHA, Bcrypt, AES, Diffie-Hellman dort nachlesen
- IT-Security ist Stoff von IT-Sicherheit (Prof. Dr. Hüttl) Grundlagen, Digitale Signaturen, Token-Based Security dort nachlesen

Probleme RESTful WebServices

- Daten werden im Klartext übertragen
  - Abhören der Internetverbindung reicht um Informationen zu bekommen
- Jeder darf alles:
  - Keine Authentifizierung des Clients
    (Benutzer bekannt?)
  - Keine Authentifizierung des Servers
    (Redet der Client wirklich mit dem richtigen Server?)
  - Keine Autorisierung
    (Darf der Benutzer diese Ressource sehen?)

Der 80% Fall!?

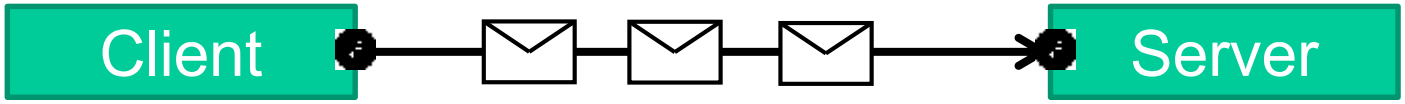
- Nutzung eines ***Webservers*** mit HTTP-Mitteln
- ***HTTPS als Protokoll*** (TLS/SSL)
  - ***= Verschlüsselte Kommunikation*** zwischen Client und Server
  - ***Port 443*** statt 80 / 8080
  - Handshake zum Kommunikationsaufbau
    - asymmetrische Verschlüsselung Austausch gemeinsames Master Secret
    - Dann: symmetrische Verschlüsselung mithilfe Master Secret
- Dann ***HTTP Basic-Auth*** bei jedem Request
  - Zum Authentisieren (und als Basis zum Autorisieren)
  - = Beweis der Identität des Benutzers / des Clients
  - Fehlschlag: HTTP-Status Code 401



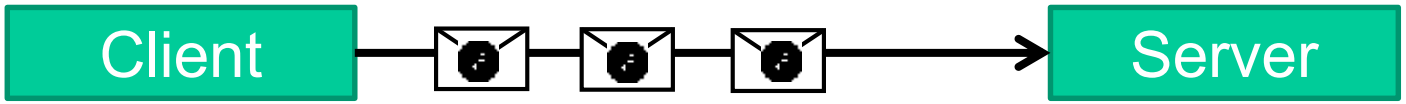
Abhörsicherheit und Man-in-the-Middle Grüße von Eve (eavesdropper)

Zwei Möglichkeiten: ***am Container konfigurierbar***

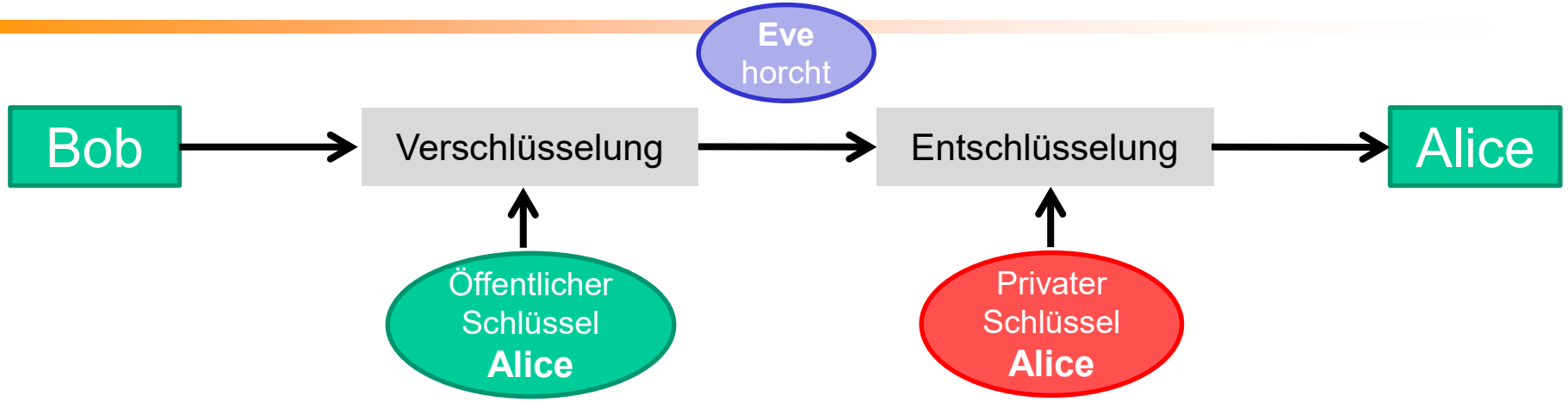
- Sicherheit auf Kanal-Ebene (***HTTPS***)
  - Verschlüsselte Übertragung (SSL/TLS)
  - Sender und Empfänger tauschen Schlüssel aus



- Sicherheit auf Paket-Ebene
  - (Teilweise) verschlüsselte Datenpakete
  - Beispiel: Verschlüsselung (und evtl. signiert) des SOAP-Body aber unverschlüsselter SOAP-Header, damit das Routing noch funktioniert



Verschlüsselung und Entschlüsselung Asymmetrische Verfahren



- Asymmetrische Verschlüsselung, z.B. mit RSA
  - Verschlüsselung mit ***Öffentlichem Schlüssel***
    (Jeder kann Alice eine Nachricht schicken)
  - Entschlüsselung mit ***Privatem Schlüssel***
    (Nur Alice kann die Nachricht entschlüsseln)
- Darüber Schlüsselaustausch für symmetrische Verschlüsselung (Diffie-Hellman-Verfahren),
- Symmetrisch ca. 1000x schneller als asymmetrisch

Historische Randnotizen

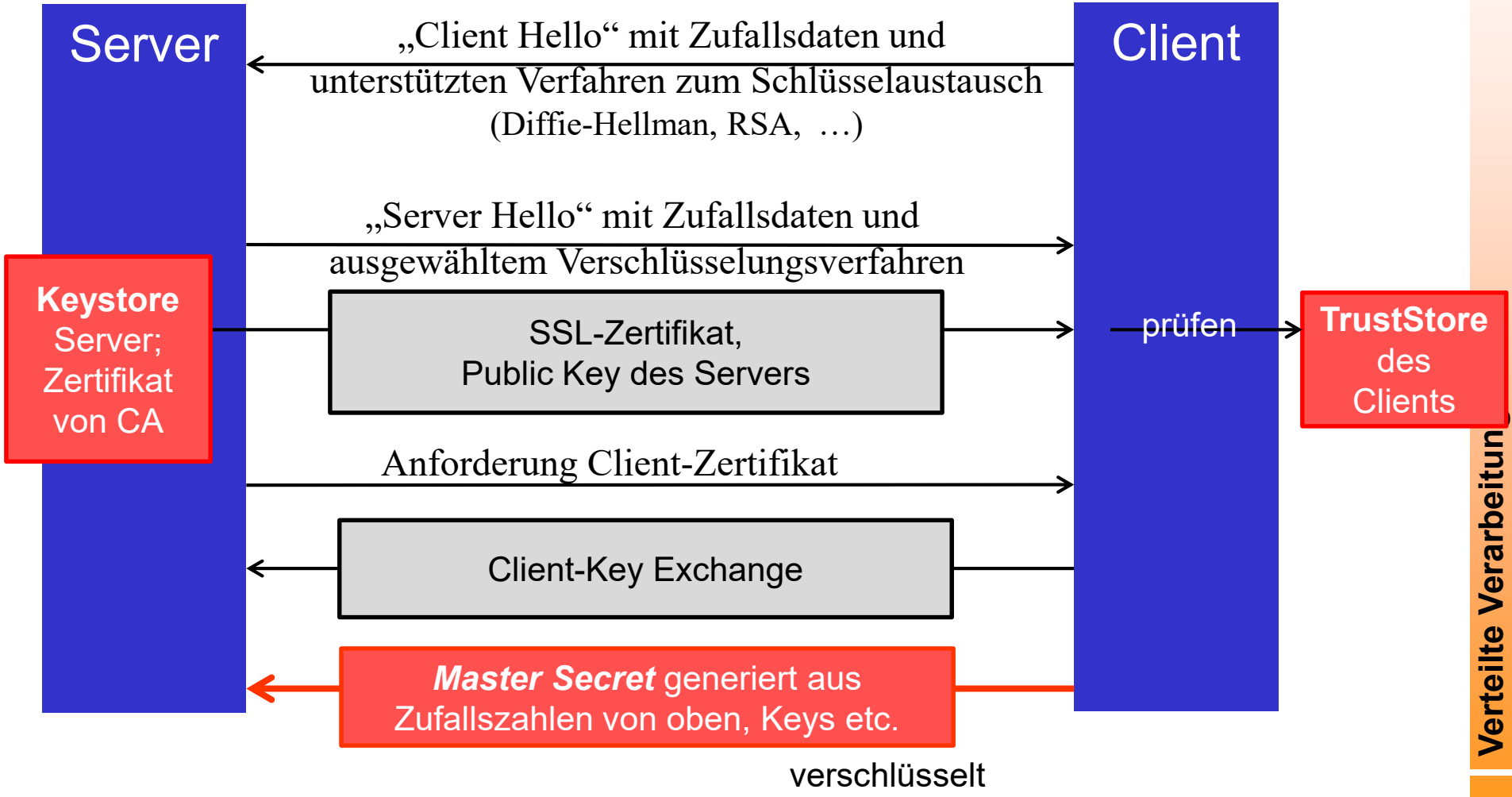
- SSL = Secure Socket Layer, Erfindung von Netscape
- SSL 3 Released in 1996 mit Netscape 1.1
  - Grundlage des modernen e-commerce
  - Vorläufer: Hatten zu viele Fehler
- Standardisiert als TLS 1.0 (Transport Level Security) in 1999 (-> Browser Wars)
- Aktuell: TLS 1.3
  - Vorteil: Sehr gute Konfigurierbarkeit und Flexibilität (Austausch sämtlicher Algorithmen, während „Client Hello“
  - Siehe: https://tools.ietf.org/html/rfc5246

HTTPS Handshake, asymmetrischer Teil

https://tools.ietf.org/html/rfc6101

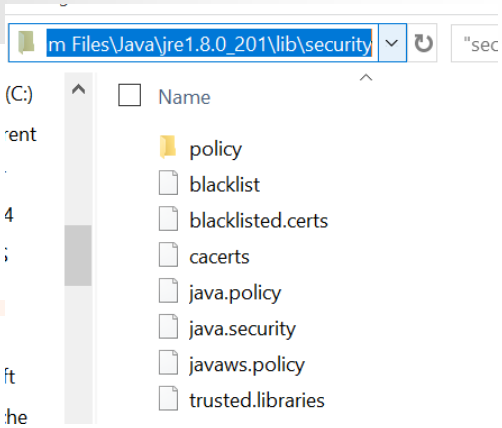
Siehe auch

https://upload.wikimedia.org/wikipedia/commons/a/ae/SSL_handshake_with_two_way_authentication_with_certificates.svg

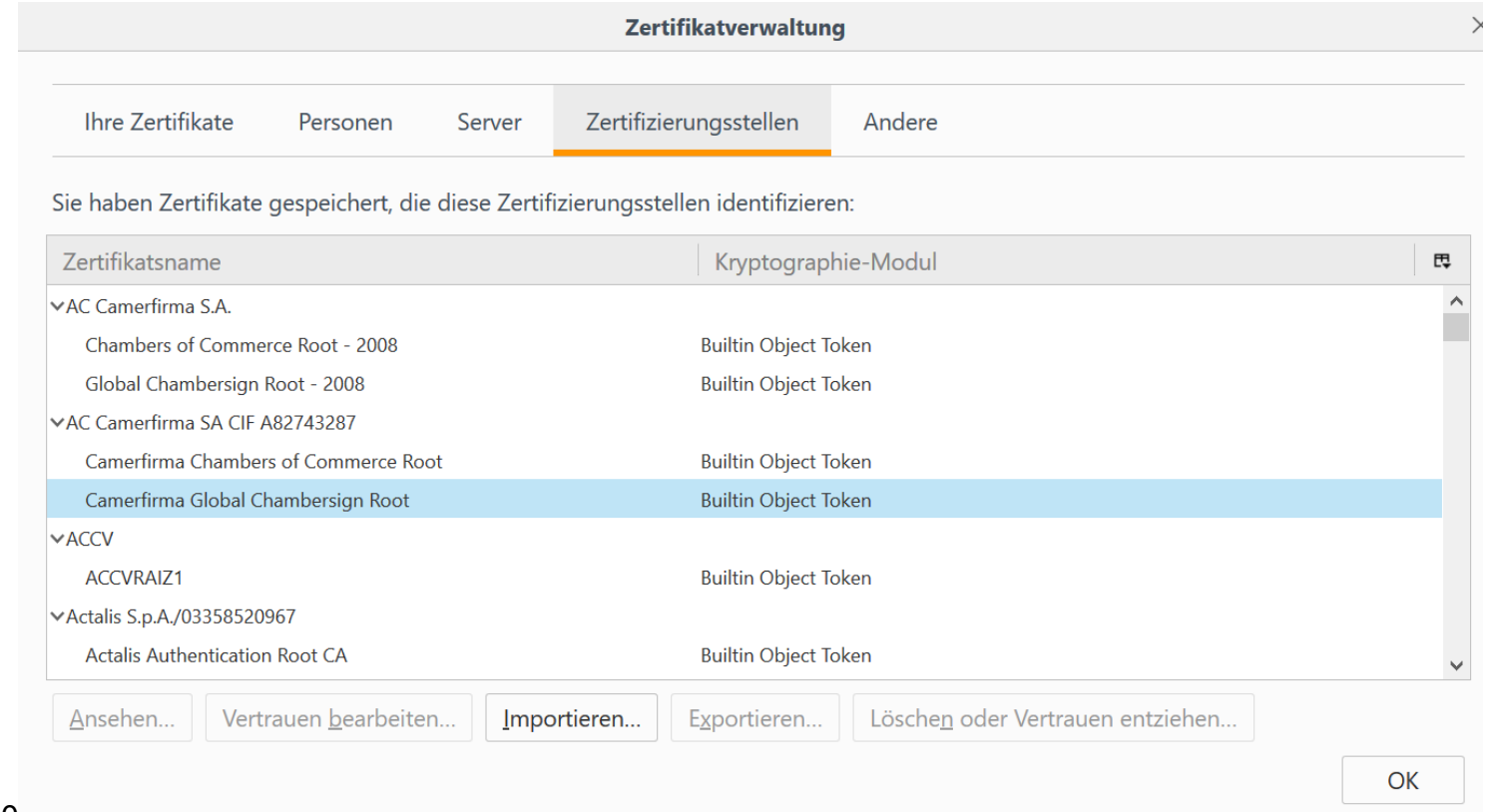


TrustStore

Java: JAVA_HOME/lib/security/cacerts



- = Datenbank mit Wurzelzertifikaten von ***vertrauenswürdigen*** Zertifizierungs-Instanzen



Details zur vereinbarten Verschlüsselung Cipher Suite

(https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4)

Cipher suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Cert. type: X.509

Hash code: -868021955

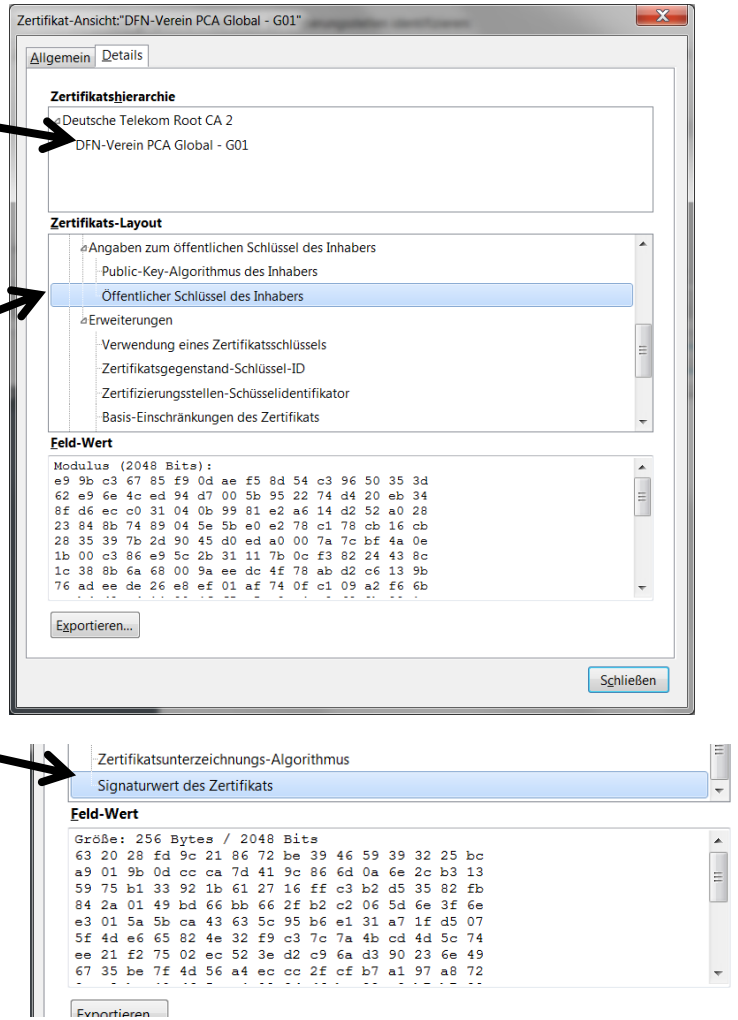
Algorithm: RSA

Format: X.509

- Erklärung (https://tools.ietf.org/html/rfc5289):
  - TLS = Transport Level Security
  - ECDHE = Elliptic Diffie-Hellman Key Exchange (Siehe Https-Grafik)
  - RSA = Rivest, Shamir, and Adleman (Public Key Algorithmus)
  - AES_128_GCM = Advanced Encryption Standard (128 Bit Schlüssellänge, Block Cypher), Galois Counter Mode
  - SHA256 = Kryptographische Hash-Funktion (= Fingerabdruck)
  - X.509 = Datenformat des Zertifikats

Asymmetrische Verschlüsselung z.B. für den SSL-Handshake

- ***Certification Authority*** beglaubigt Digitales Zertifikat
  (sonst könnte man so tun, als wäre man jemand anders
  = Man in the Middle)
- ***Privater Schlüssel*** zum Verschlüsseln der Nachricht
  Ist nicht enthalten!!!
- ***Öffentlicher Schlüssel*** zum Entschlüsseln der Nachricht
  (PKCS #1 RSA-Verschlüsselung)
- ***Digitales Zertifikat*** = Öffentlicher Schlüssel + Digitale Unterschrift
  (Format z.B. X.503 v3)

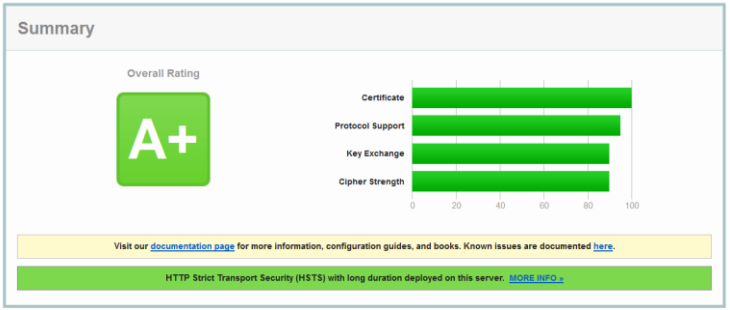


Beispiel: SSL-Zertifikat des innovationlabs

https://www.ssllabs.com/ssltest/analyze.html?d=innovationlab.fh-rosenheim.de

Server Key and Certificate #1

Subject	innovationlab.fh-rosenheim.de Fingerprint SHA1: 37a621d0b35f84faefbf803712fa20ea7465058e Pin SHA256: cH05wlC4UYBRBXseL4rp7/57g9SKxPhvs0v3sqYSSCE=
Common names	innovationlab.fh-rosenheim.de
Alternative names	innovationlab.fh-rosenheim.de
Valid from	Tue, 26 Apr 2016 13:12:09 UTC
Valid until	Tue, 09 Jul 2019 23:59:00 UTC (expires in 3 years)
Key	RSA 2048 bits (e 65537)
Weak key (Debian)	No
Issuer	FH-RO CA - G02 AIA: http://cdp1.pca.dfn.de/fh-rosenheim-ca/pub/cacert/g_cacert.crt AIA: http://cdp2.pca.dfn.de/fh-rosenheim-ca/pub/cacert/g_cacert.crt
Signature algorithm	SHA256withRSA
Extended Validation	No
Revocation information	CRL, OCSP CRL: http://cdp1.pca.dfn.de/fh-rosenheim-ca/pub/crl/g_cacrl.crl OCSP: http://ocsp.pca.dfn.de/OCSP-Server/OCSP
Revocation status	Good (not revoked)
Trusted	**Yes**



HTTPs

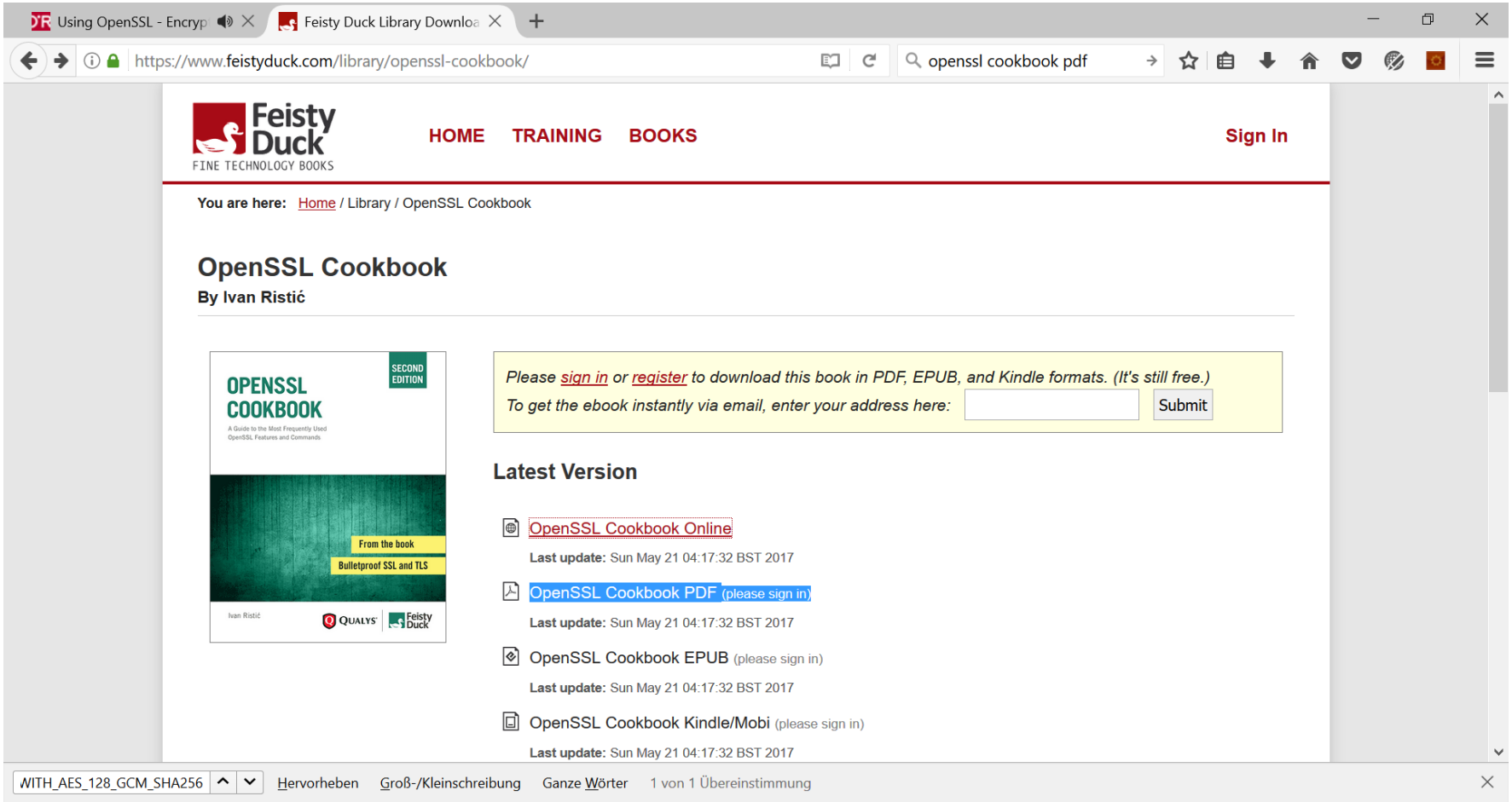

Client
Internet
https, user + pwd
WebServer
(Reverse Proxy)
http, user
Application

Was muss der Administrator tun?

- (Web) Server muss ein Zertifikat besitzen
  - Self-Signed Certificate (selber generiert, macht Probleme)
  - Zertifikat von öffentlicher CA (kostet)
- Wichtig: Public Key und Private Key = Schlüssel Paar (da der Server der Dienstanbieter ist)
- Zertifikat (Schlüssel Paar) wird im „keystore“ (Datei, irgendwo am Server) gespeichert.
- Server wird so konfiguriert, dass er das Zertifikat aus dem keystore verwendet
- Hinweise bei Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS

Werkzeug zur Schlüsselerzeugung

OpenSSL: https://www.feistyduck.com/library/openssl-cookbook/



HTTPS / SSL

Welche Angriffe sind nicht mehr möglich?

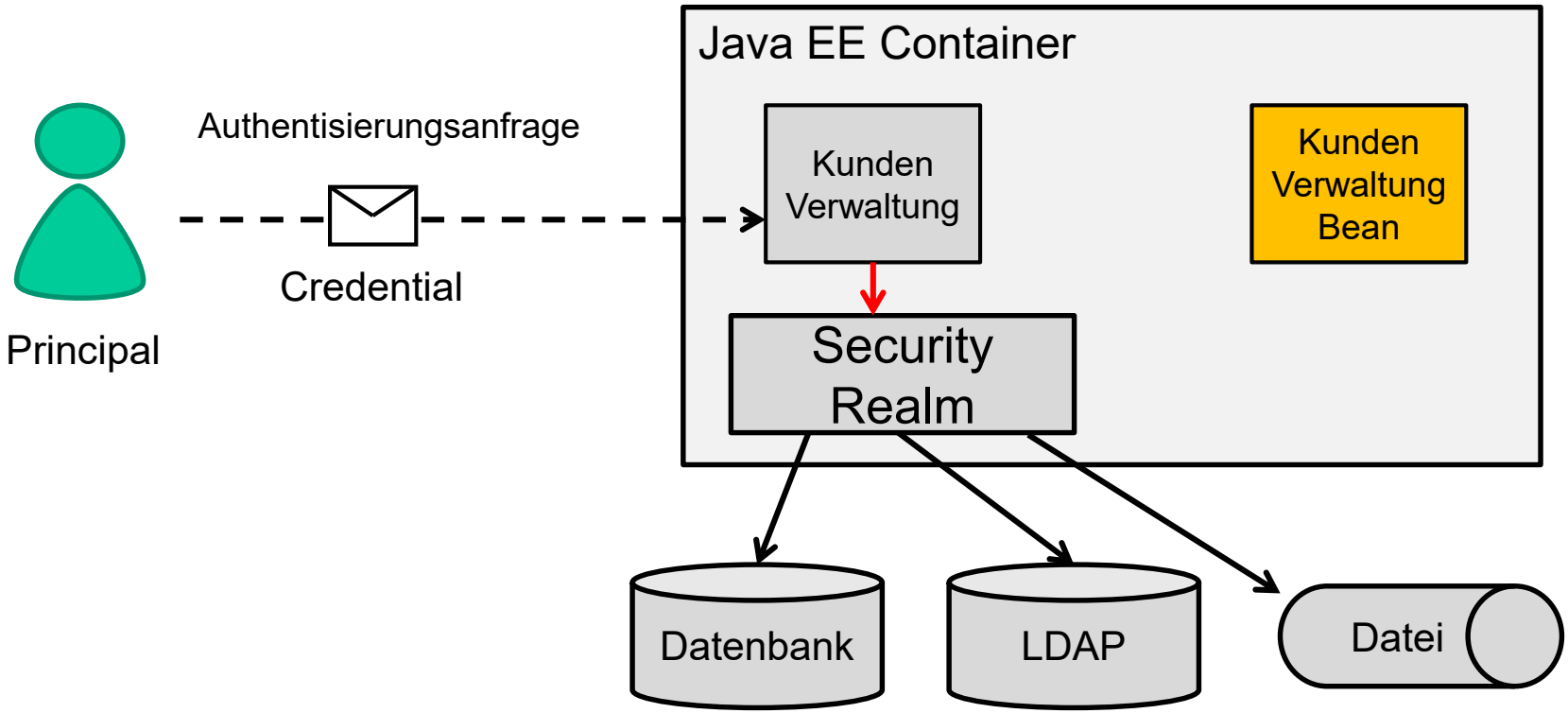
- Client redet mit falschem Server
  - Server ***beweist Identität*** gegenüber Client mit SSL-Zertifikat (Public Key)
  - SSL-Zertifikat von einer ***vertrauenswürdigen Zertifizierungsstelle*** (dfn, …)
  - Bei selbst erstellten Zertifikaten: Probleme im Browser
- Abhören: Bei etablierter Verbindung kein Angriff durch Abhören der Kommunikation möglich (man in the middle), da Kommunikation verschlüsselt

Welche Angriffe gehen noch?

- Fake Client noch möglich: Client kann dem Server nicht die Identität beweisen (dazu Authentifizierung).
- Brute-Force Angriff auf Verschlüsselung, dauert lange
  Dauer: siehe Skriptum Schmidt / Hüttl
- Cross-Site-Request-Forgery

Security

= Authentisierung und Autorisierung



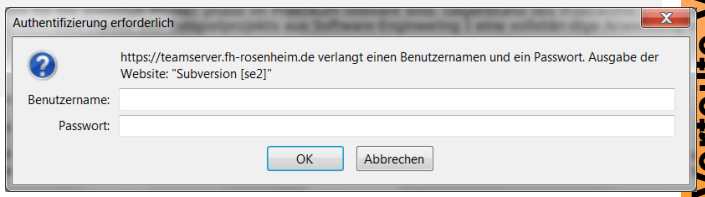
Authentisierung und Autorisierung

- ***Authentisierung*** = Absichern, dass der Benutzer der ist, der er vorgibt zu sein
  - Technologien: HTTP-Basic (für Webseiten), LDAP (als Schnittstelle zum zentralen Berechtigungssystem), OpenID (Single Sign On), X.509
  - Wichtig: Integration zentraler Berechtigungs-System in Ihre Software (Nicht jede Software sollte selbst Benutzer verwalten)
- ***Autorisierung*** = Darf der Benutzer diese Funktion aufrufen bzw. diese Daten sehen / ändern?
  - Technologien: z.B. Access Control Lists (ACL)
  - Wichtig: Berechtigungen zentral verwalten
- Flexibilität im Einsatz der Berechtigungsprüfung
  - Schnittstellen zu zentralen Systemen
  - Anpassbarkeit des Verhaltens an Anforderungen des Systems

Der 80% Fall?

Siehe Skriptum GDI 1, Prof. Jochen Schmidt
Siehe Skriptum IT-Sicherheit, Prof. Reiner Hüttl

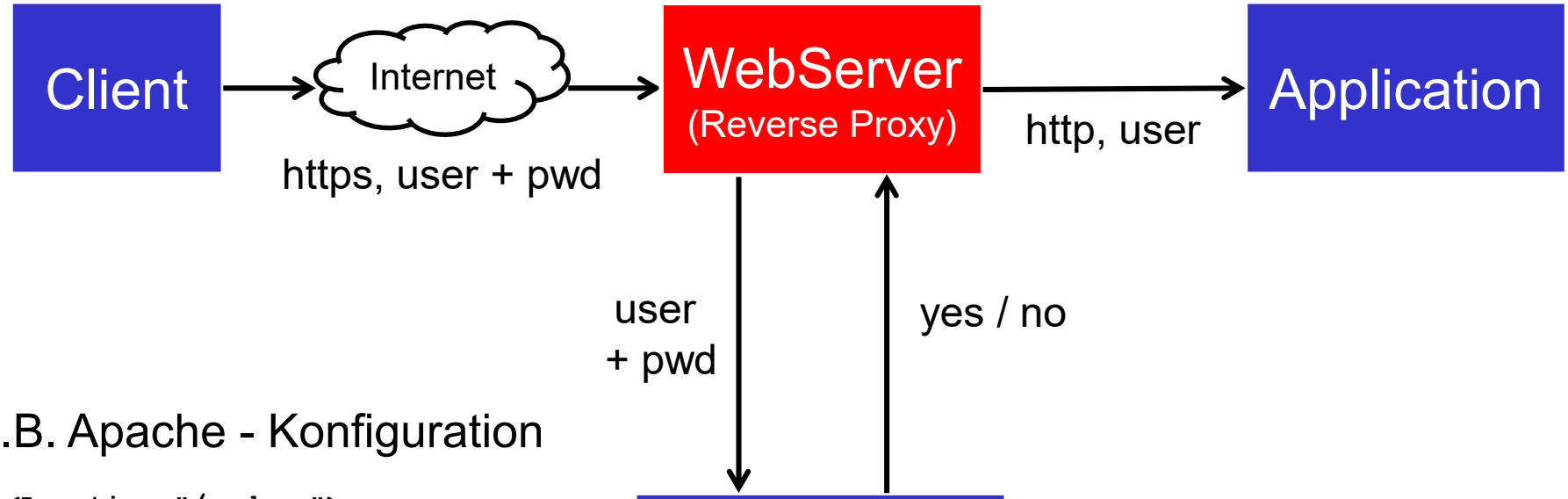
- Nutzung eines ***Webservers*** mit HTTP-Mitteln
- ***HTTPS als Protokoll*** (TLS/SSL)
  - = Verschlüsselte Kommunikation zwischen Client und Server
  - Port 443 statt 80 / 8080
  - Handshake zum Kommunikationsaufbau
    - asymmetrische Verschlüsselung Austausch gemeinsames Master Secret
  - Dann: Symmetrische Verschlüsselung mithilfe Master Secret
- Dann ***HTTP Basic-Auth*** bei jedem Request
  - = Beweis der Identität des Benutzers / des Clients
  - Fehlschlag: HTTP-Status Code 401



HTTP-Basic Auth

- Client verwendet Benutzernamen und Passwort
- Base 64 Codierung von: Benutzername:Passwort
- Im HTTP-Header übertragen (Authorization Header)
- Achtung: Base 64 ist praktisch Klartext, daher kein Basic Auth ohne HTTPS (!), Alternative: Digest-Auth

HTTPs + Basic Auth



z.B. Apache - Konfiguration

```
<Location "/orders">
  SSLRequireSSL
  AuthName "OrderManagement"
  AuthType Basic
  AuthUserFile /etc/apache2/users
  AuthGroupFile /etc/apache2/groups
  Require group internal
</Location>
```

Security in Spring Boot

- Umfangreiche Default Implementierung
- Default = Basic Auth mit „user“ + „generiertem Passwort“ (siehe Logfile)

- POM:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Beispiel mit Spring Security Framework
// Account-Daten Speichern (RDBMS)

```
@RestController
public class GreetingsController {

   @RequestMapping(method= RequestMethod.GET, value = "/greeting")
   public Map<String, String> greetings(Principal p) {
      return Collections.singletonMap("content", "Hello "
            + p.getName());
   }
}
```

In POM-Datei “anschalten”

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Eigene Account-Verwaltung (nur im Notfall)

Beispiel mit Spring Security Framework
// Account-Daten Speichern (RDBMS)

```
@Entity
public class Account {
   @Id
   @GeneratedValue
   private Long id;
   private String username;
   private String password; // Klartext ????
   private boolean isActive;
// …
}

public interface AccountRepository
        extends CrudRepository<Account, Long> {

    public Optional<Account> findByUsername(String username);
}
```

Beispiel mit Spring Security Framework

Hierzu wird ein Account User Details Service bereit gestellt

```
public class AccountUserDetailsService implements UserDetailsService {
    private final AccountRepository repo;

    @Autowired
    public AccountUserDetailsService(AccountRepository repo) {this.repo = repo;}

    @Override
    public UserDetails loadUserByUsername(String username)
                                throws UsernameNotFoundException {
        Account accout = repo.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException(
                        "User " + username + " not found"));


        UserDetails details = User.builder()
                .username(accout.getUsername())
                .password(accout.getPassword())
                .roles("ADMIN","USER")
                .build();
        return details;
    }
}
```

UserDetailsService in Authentisierung einhängen

```
@Configuration
@EnableWebSecurity
public class TauschSecurityConfig extends WebSecurityConfigurerAdapter {
    private final AccountUserDetailsService accountUserDetailsService;

    @Bean
    @Override
    protected UserDetailsService userDetailsService() {
        return accountUserDetailsService;
    }

    @Bean
    public AuthenticationProvider authProvider () {
        DaoAuthenticationProvider daoProvider
            = new DaoAuthenticationProvider();
        daoProvider.setUserDetailsService(accountUserDetailsService);
        daoProvider.setPasswordEncoder(new BCryptPasswordEncoder());
    return daoProvider;
}
```

Leider noch für alle anderen Request-Typen konfigurieren

```
@Configuration
@EnableWebSecurity
public class TauschSecurityConfig extends WebSecurityConfigurerAdapter {
    private final AccountUserDetailsService accountUserDetailsService;

@Override
    protected void configure(HttpSecurity http) throws Exception {
        http.headers().frameOptions().disable();  // H2-Console
        http.csrf().disable()
                .authorizeRequests()
                .anyRequest().fullyAuthenticated()
                .and()
                .httpBasic();
    }

}
```

HTTP Digest Authentication als Alternative

- Basic Auth: Passwort wird im Klartext übertragen
- Idee Digest Auth: Übertragung des Passworts im Klartext vermeiden:
  - Server überträgt „nonce“ (Zufallswert) beim ersten Aufruf
  - Client bildet aus nonce, Benutzernamen, Passwort und weiteren Informationen einen MD5-Hashwert
    `MD5(MD5(Benutzername:Realm:Passwort):nonce:MD5(http-Methode:URI))`
  - Client überträgt nun nonce, sowie die anderen Informationen im Klartext (bis auf Passwort) zusätzlich den MD5-Hashwert
  - Server rechnet mit den gegebenen Informationen nun selbst den MD5-Hashwert aus und vergleicht damit den übertragenen.

Möglichkeiten zur Authentisierung und für SSO Dieses ist die REGEL bei ihren Anwendungen

- Sie wollen sicher keine neue Benutzerverwaltung für jede Anwendung bauen! (Wenn sie schon eine haben)
  - Daher: Vorhandene Benutzerverwaltung anbinden, LDAP, Oauth 2, …
  - Häufig Token-Basierte Verfahren
- HMAC (wie bei AWS oder Google Code)
- Open ID, ***OAuth 2.0 mit JWT/SAML***

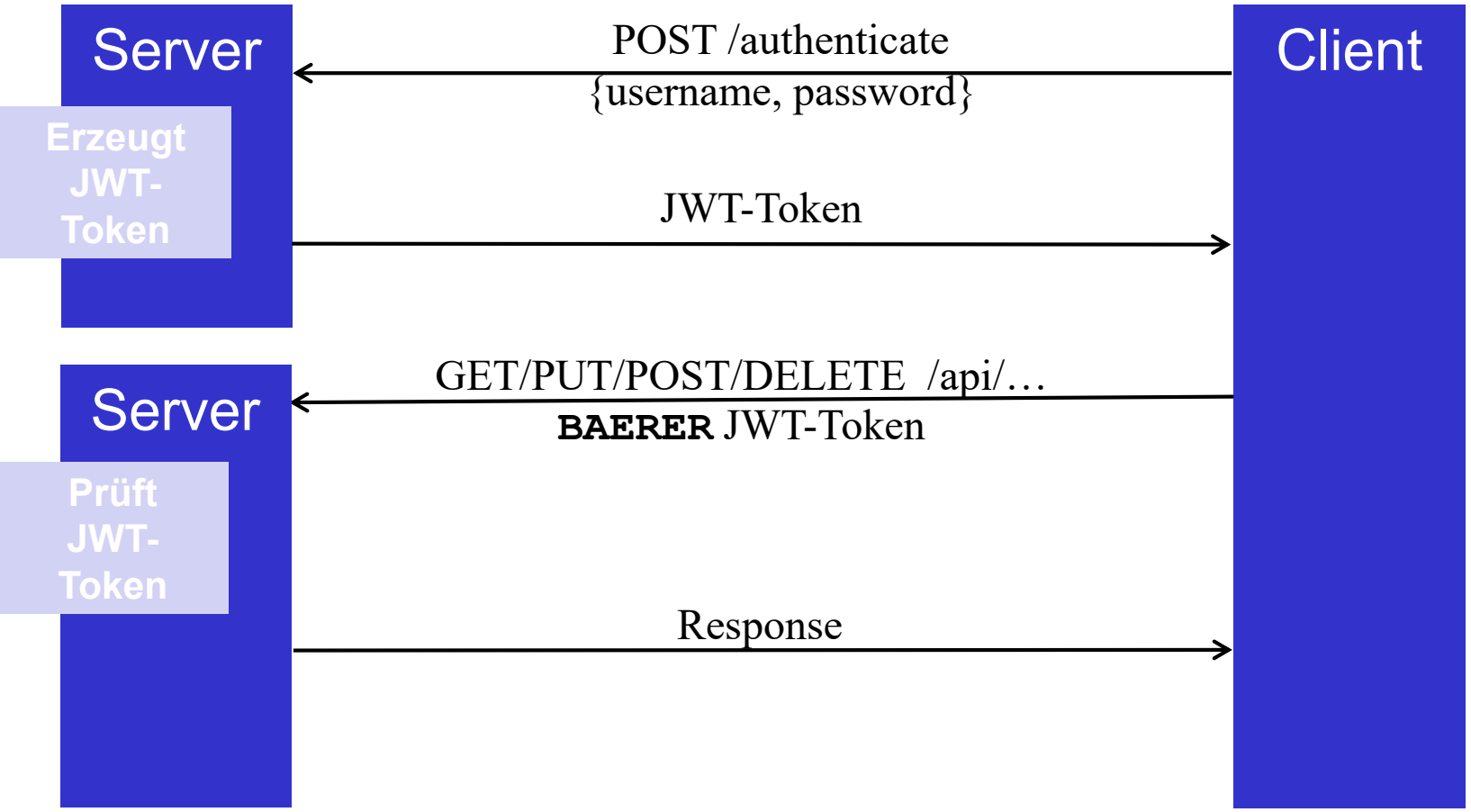
- Details Siehe weiterführende Veranstaltungen (CA, CC und andere)

Tokenbasierte Authentisierung mit JWT

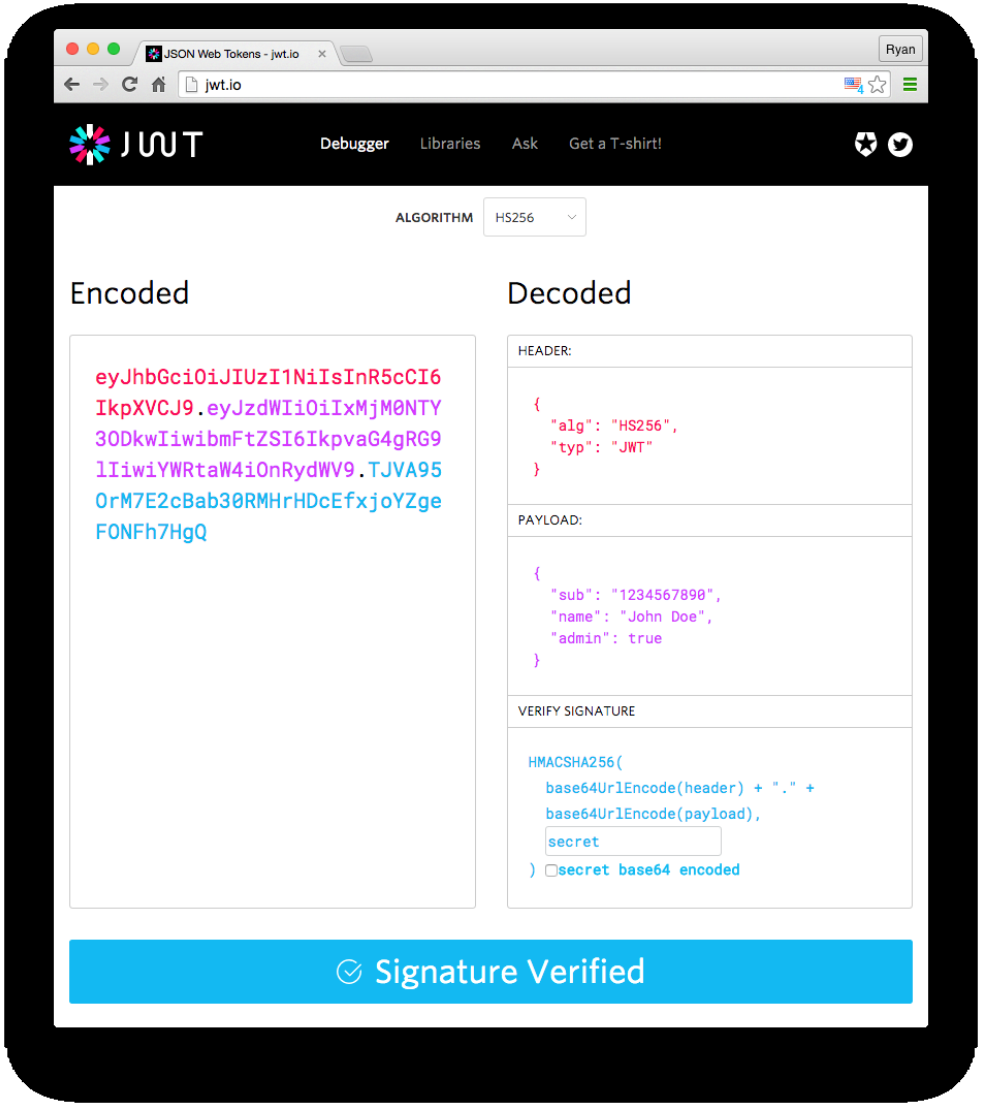
JWT Inhalte

- Header
  - Unterschriftsverfahren
  - Typ: JWT
- Payload
  - Informationen zu Claims
  - z.B. Nutzername, Berechtigungen, Datum Gültigkeit
- Signatur
  - Schützt Integrität des Inhalts
  - HMAC + SHA256 + Geheimnis

JWT - Ablauf


Server
Erzeugt JWT-Token
Client
POST /authenticate
{username, password}
JWT-Token
Server
Prüft JWT-Token
GET/PUT/POST/DELETE /api/…
BAERER JWT-Token
Response
verschlüsselt

Beispiel: jwt.io



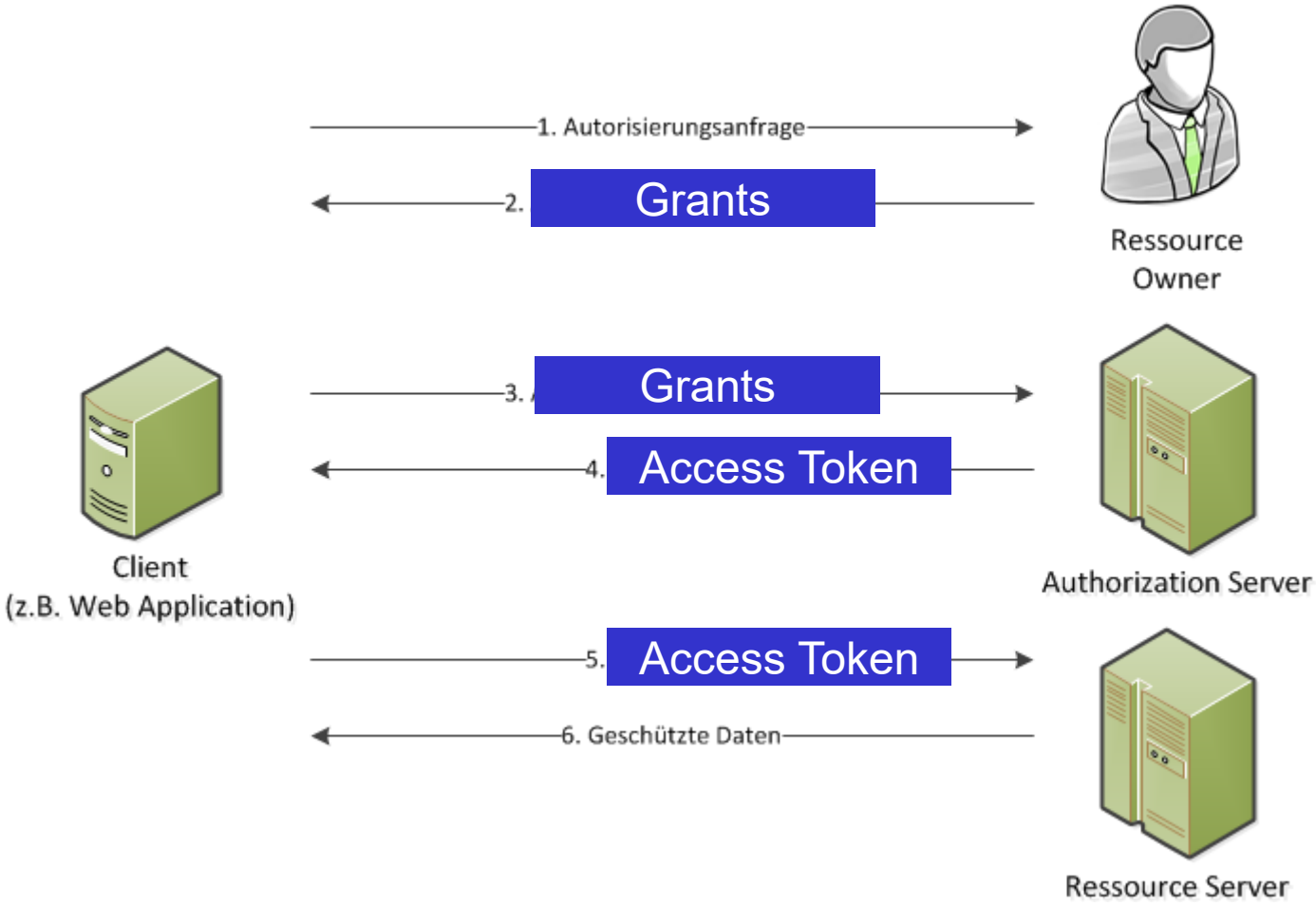
OAuth2

OAuth 2 Rollen

- Anwender (Ressource Owner)
  - Besitzt Ressourcen auf dem Ressourcen Server
  - Hat Account auf dem Authorization Server
  - Verwendet Client, der Login-Dialog des Authorization Servers zeigt
- „Client“ –Software (eigentlich Server)
  - Software z.B. von ihnen
  - Ist am Authorization Server bekannt
  - Client und Authorization Server haben Credentials ausgetauscht
- Authorization Server
  - Client wurde hier früher schon angemeldet: Beispiel Spotify (Client) ist bei Facebook registriert, daher kann man sich mit dem Facebook Account bei Spotify anmelden
  - Gibt Client auf Anfrage „Login-Dialog für Anwender“ bei Erfolg: Authorization Grant (Zugiffsrechte)
  - Client kann mit Grant dann zugriff auf Ressourcen anfragen, Bei erfolgreicher Anfrage -> Access Token für Client
- Ressource Server
  - Akzeptiert die AccessTokesn vom Authorization Server

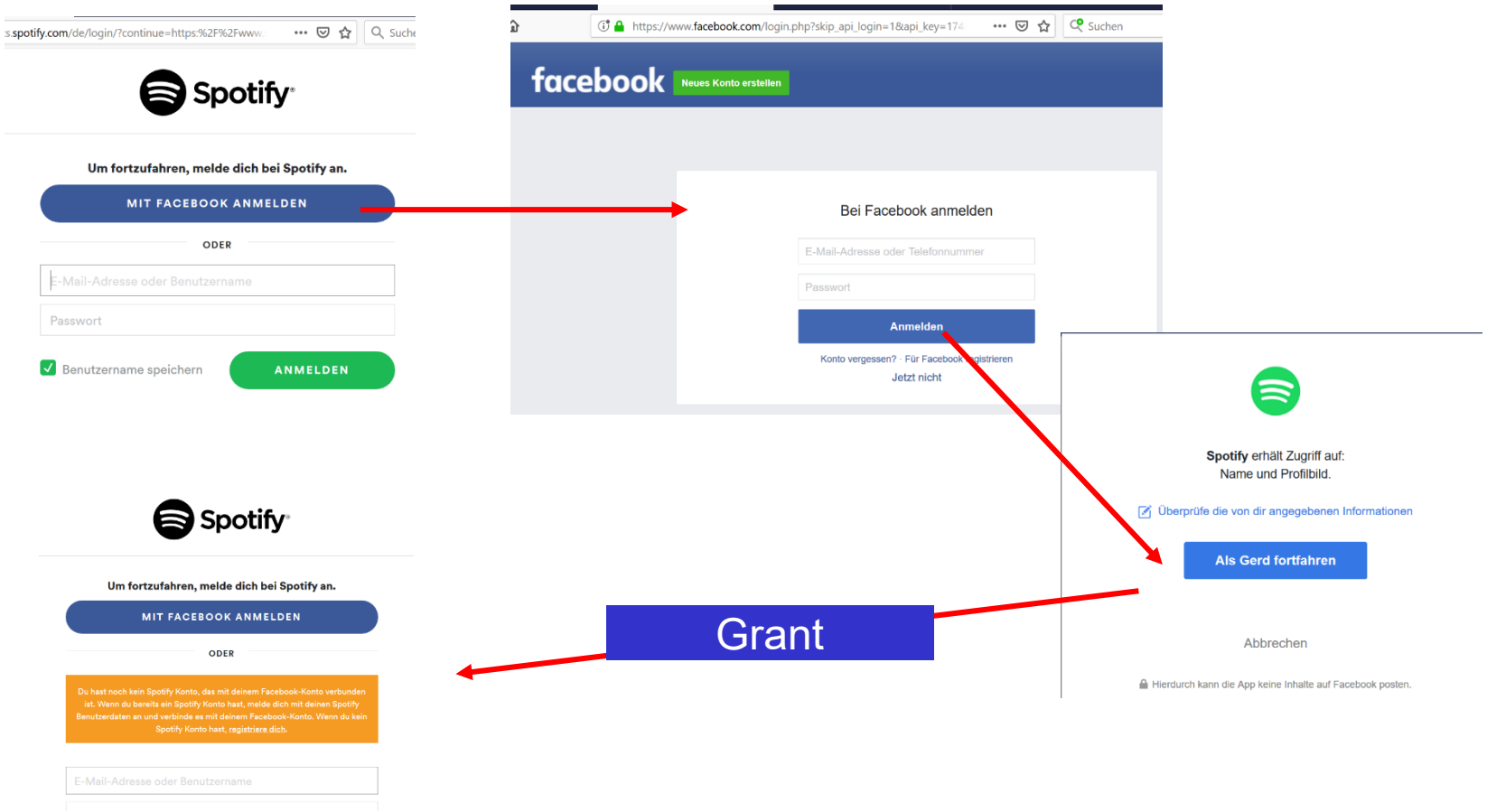
Ablauf in OAuth 2.0

Grafik aus: https://de.wikipedia.org/wiki/OAuth



OAuth 2.0: https://tools.ietf.org/html/rfc6749

Beispiel


Spotify
Um fortzufahren, melde dich bei Spotify an.
MIT FACEBOOK ANMELDEN
ODER
E-Mail-Adresse oder Benutzername
Passwort
Benutzername speichern
ANMELDEN
facebook
Neues Konto erstellen
Bei Facebook anmelden
E-Mail-Adresse oder Telefonnummer
Passwort
Anmelden
Konto vergessen? · Für Facebook registrieren
Jetzt nicht
Spotify erhält Zugriff auf:
Name und Profilbild.
Überprüfe die von dir angegebenen Informationen
Als Gerd fortfahren
Abbrechen
Hierdurch kann die App keine Inhalte auf Facebook posten.
Grant
Spotify
Um fortzufahren, melde dich bei Spotify an.
MIT FACEBOOK ANMELDEN
ODER
Du hast noch kein Spotify Konto, das mit deinem Facebook-Konto verbunden ist. Wenn du bereits ein Spotify Konto hast, melde dich mit deinen Spotify Benutzerdaten an und verbinde es mit deinem Facebook-Konto. Wenn du kein Spotify Konto hast, registriere dich.
E-Mail-Adresse oder Benutzername

Single Sign On (wichtig für Microservices): JSON Web Token (JWT), RFC 7519

- = Authentifizieren im Netz über Identity Provider (wie z.B. Facebook, LinkedIn und andere)
  - Benutzer Informationen im Token enthalten, gespeichert in sog. Claims
  - Sonst ggf. Aufruf des Identity Providers bei jedem Request
- Standardisiertes Token-Format
  - = Header, Payload, Signatur
  - Verschlüsselt
  - Ggf. Signiert
- Kann für Single Sign On verwendet werden
  - Grant Type (= Zugriffsgenehmigung) im Rahmen OAuth 2.0

Autorisierung

Programmatische Autorisierung: JSR 250 Annotationen aus javax.security

- JSR 250 definiert Annotationen zur Definition von Autorisierungsinformationen
- Beispiele: `@RolesAllowed`, `@DenyAll`, `@PermitAll`, `@DeclareRoles`, `@RunAs`
- Nutzerinformationen im Rest-Aufruf enthalten (z.B. wegen Basic Auth)

```
@Path("/kunden")
@PermitAll
public class KundenverwaltungRessource {
   @GET @Produces(MediaType.APPLICATION_JSON)
   @RolesAllowed({"user","admin"})
    public List<Kunde> kundenListe() {
       return new ArrayList<Kunde>( kunden.values());
    }
```

Password und Content Security

Passwörter Speichern
[Wenn sie doch selber die User verwalten]

- Idee: Passwort ***verschlüsseln*** mit AES/DES …
  - Unüblich, da wir eigentlich Passwörter nicht entschlüsseln müssen (siehe unten)
  - Verfahren angreifbar -> Servercode Hacken -> AES Schlüssel erbeuten -> alle Passwörter erbeuten
- Zweite Idee: ***Hash-Verfahren*** (MD5, SHA256, BCrypt)
  - Hash-Funktion rechnet aus Passwort Hash-Wert mit konstanter länge aus unabhängig von der Länge des Passworts
  - Bei Login wird der Hash des eingegebenen Passworts mit dem gespeicherten Hash verglichen
  - MD5 und SHA sind optimiert auf große Datenmengen (vgl. z.B. git verwendet SHA1) und Performance
  - Daher: ***BCrypt*** als spezieller Hash-Code für Passwörter

Passwort verschlüsseln

```
@Configuration
@EnableWebSecurity
public class TauschSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public AuthenticationProvider authProvider () {
        DaoAuthenticationProvider daoProvider
            = new DaoAuthenticationProvider();
        daoProvider.setUserDetailsService(accountUserDetailsService);
        daoProvider.setPasswordEncoder(new BCryptPasswordEncoder());
        return daoProvider;
    }
```

select * from account;

ID	IS_ACTIVE	PASSWORD	USERNAME
4	TRUE	$2a$10$5v3cB2fSY71A1qotafW2fuXpMU/2AkjYWb.l/UVQo1VgEMtzpuMe.	gerd
5	TRUE	$2a$10$wWSsMQHof3lqPfUFjv3Lf.hHrXwXHueolvjr2ZQGFUKKxoJw59gPK	hinz
6	TRUE	$2a$10$cBiHmI08lv5Xue5YyvjWt./t4gMLCkTQqEyT6J9RoRDPlC9wzKESm	bonny

(3 rows, 6 ms)

Edit

Inhalte verschlüsseln

```
public class UserNameConverter implements AttributeConverter<String, String>
{
    private static final String ALGORITHM = "AES/ECB/PKCS5Padding";
    private static final byte[] KEY = "GerdsSuperSecret".getBytes();

    @Override
    public String convertToDatabaseColumn(String entityData) {
        Key key = new SecretKeySpec(KEY, "AES");
        try {
            Cipher c = Cipher.getInstance(ALGORITHM);
            c.init(Cipher.ENCRYPT_MODE, key);
            return Base64.getEncoder().
                    encodeToString(
                        c.doFinal(entityData.getBytes()));
        } catch (Exception e) { throw new RuntimeException(e);
        }
    }
```

Inhalte entschlüsseln

```
    @Override
    public String convertToEntityAttribute(String dbData) {
        Key key = new SecretKeySpec(KEY, "AES");
        try {
            Cipher c = Cipher.getInstance(ALGORITHM);
            c.init(Cipher.DECRYPT_MODE, key);
            return new String(
               c.doFinal(Base64.getDecoder().decode(dbData)));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

Literatur

- Tilkov: REST und HTTP, dpunkt, 3. Auflage, 2009

- Richardson, Ruby: RESTful Web Services, O‘Reilly 2007

- Burke: RESTful Java, O‘Reilly 2009