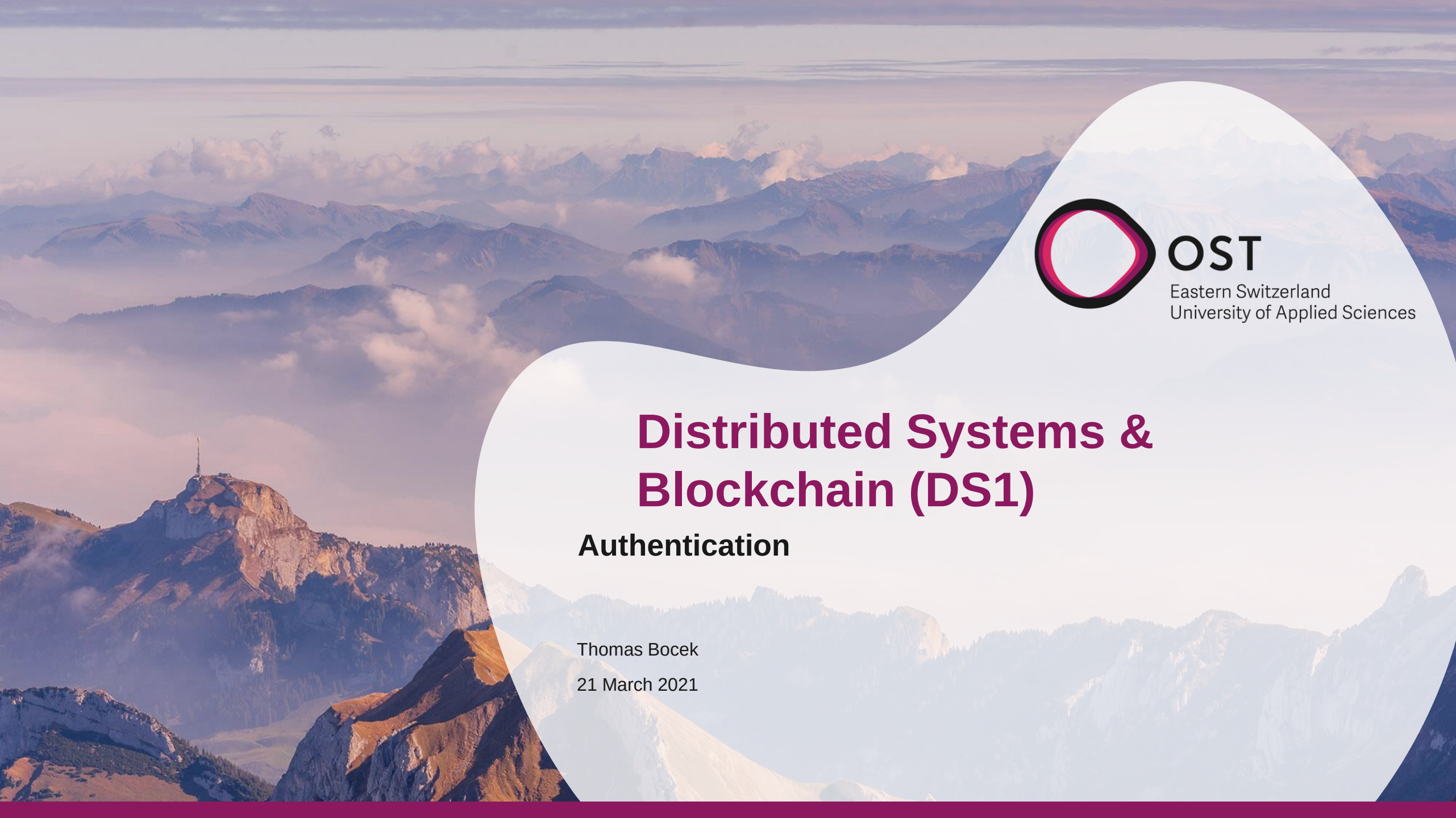


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems & Blockchain (DS1)

Authentication

Thomas Bocek

21 March 2021

Information security – Key concepts / Access control

- Confidentiality
  - Confidentiality protects transmitted data against eavesdroppers
- Integrity
  - Provides protection against the modification of a message
- Availability
  - Data needs to be available when needed
- Non-repudiation
  - Non-repudiation provides that neither the sender nor the receiver can deny that a communication has taken place
- Identification
  - E.g. with a username “alice”, you are claiming to be Alice
- Authentication
  - Verifying a claim of identity. E.g., Alice shows passport, so an other person can authenticate her. Different authentication types:
    - Something you know: things such as a PIN, a password
    - Something you have: a key, a swipe card
    - Something you are: biometrics: palm, fingerprint
- Authorization
  - Determined what resources they an authenticated user is permitted to access

OST

Authentication

- Authentication
  - Single-factor authentication
    - E.g. password
  - Multi-factor authentication / 2FA
    - E.g. password **and** software token, SMS (15.03.2021)
- Password rules
  - Don’t use:
    - The name of a pet, child, family member, or significant other
    - Anniversary dates and birthdays
    - Birthplace
    - Name of a favorite holiday
    - Something related to a favorite sports team
    - The word "password“
  - Don’t’ reuse passwords, use password managers

- Don’t enter passwords on unencrypted sites
- Password length: password cracking with 5000$ in 2018 with hashcat
  - Hashtype: WPA/WPA2: 1190.5 kH/s

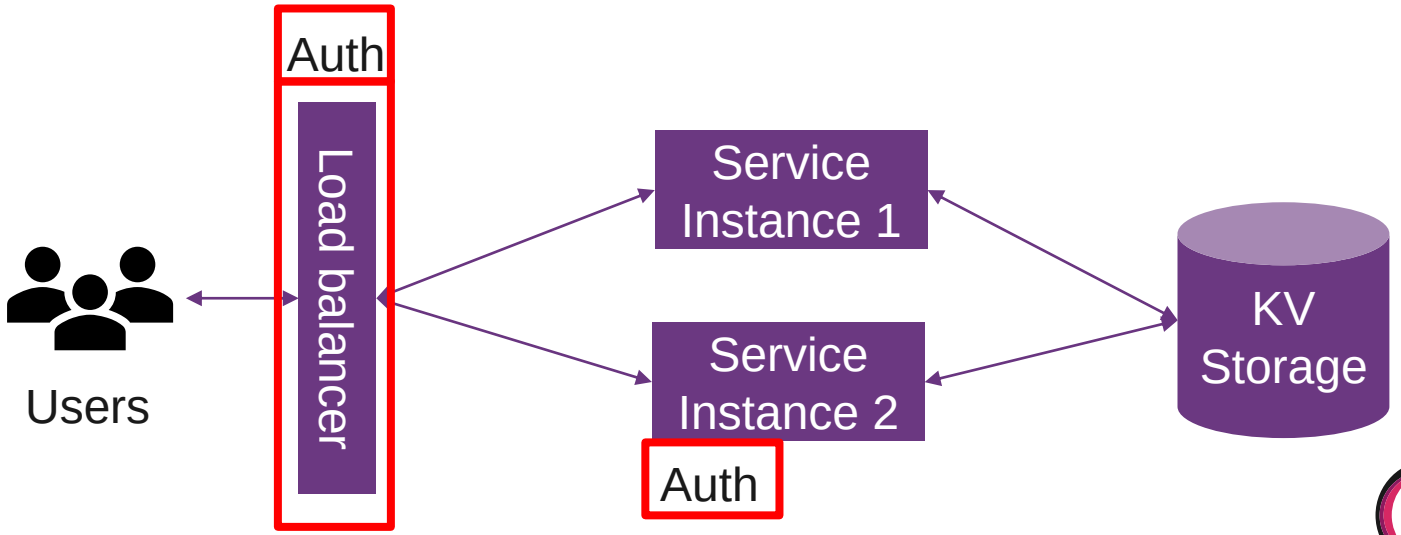
Pw length	Combinations	Time
6	11m	9s
7	656m	9m
8	38b	8h
9	$7 * 10^{15}$	186y
10	$4 * 10^{17}$	11ky
11	$2 * 10^{19}$	665ky
12	$1 * 10^{21}$	38my

- Combinations depend on PW complexity

OST

Authentication

- Software token: TOTP (Time-based One-time Password)
  - Often used as 2nd factor, e.g., in andOTP
  - Based on keyed-hash message authentication code
  - ~ hash(key + message)
  - TOTP(K, T) Truncate(HMAC-SHA-256(K,T))
  - K=shared secret
  - T=Current Unix time / X
    - (X default is 30sec)
- Challenge Task
  - In service, e.g., your HTTP server
  - In load balance, e.g. traefik, jwt
  - Choices…



Authentication

- Basic Auth
  - Load balancer
  - Services (keep state! E.g., userlist)
- Basic Auth, only with HTTPS
  - Logout: use wrong credentials – inconsistent behavior
  - Client will provide
    - Authorization: Basic <base64>
    - <base64> contains the username:password in base64
    - echo -n "dGVzdDp0ZXN0" | base64 -d
  - Can be encode in URL
    - https://username:password@dsl.hsr.ch
    - Attention:
    - http://www.google.com:search@evil.com
- Server will reply with header
  - WWW-Authenticate: Basic realm=“restricted area“
  - The user will see the information “restricted area”

OST

Authentication

- Digest Auth
  - Also available in traefik
  - Hash + nonce, against replay attacks
  - Server sends
    - WWW-Authenticate: Digest realm="testrealm@host.com",

      …

      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",

      …
  - Client sends in HTTP header
    - Authorization: Digest username="Alice",

      realm="testrealm@host.com",

      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",

      uri="/dir/index.html",

      …

      response="6629fae49393a05397450978507c4ef1"

- Advantages
  - PW not in clear text (MD5), can be "SHA-256", "SHA-256-sess", "SHA-512" and "SHA-512-sess“
    - sess: "session key" for "authentication session"
  - Nonce for replay protection for client and server
- Disadvantages
  - Browser L&F
  - Cannot use scrypt or bcrypt to store PWs

OST

Authentication

- Public/private key
- Create SSL CA certificates for server with openssl command

```
# This will return a 403 to all clients without a proper certificate
if ($ssl_client_verify != "SUCCESS") { return 403; }

# This tells Nginx what CA to verify against
ssl_client_certificate /tmp/ca.pem;

# This tells Nginx to verify clients
ssl_verify_client on;
```

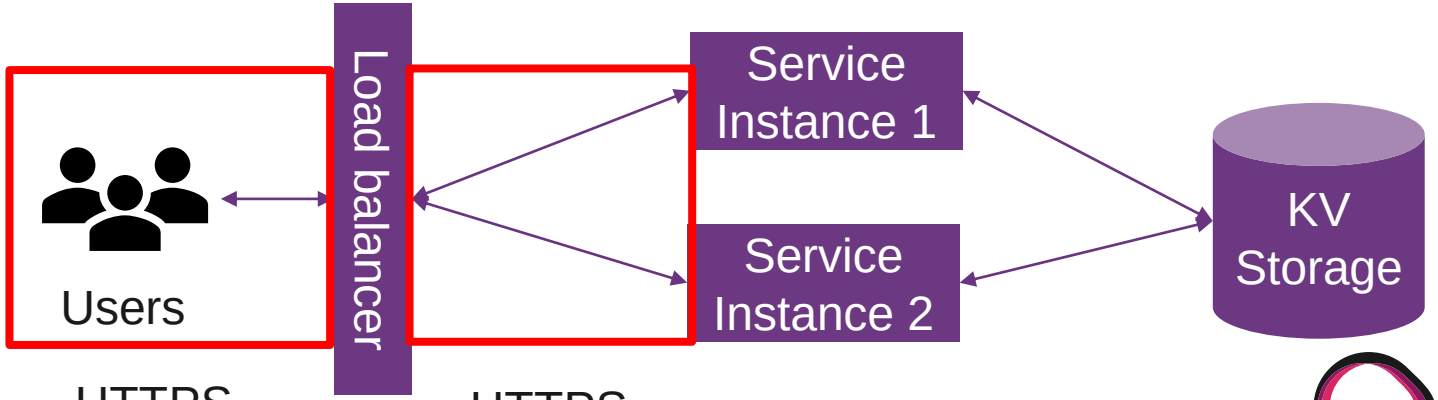
- Create CA

```
#generate CA certificate and private key
openssl req -new -nodes -x509 -keyout ca.key -
newkey rsa:4096 -out ca.pem -days 3650
```

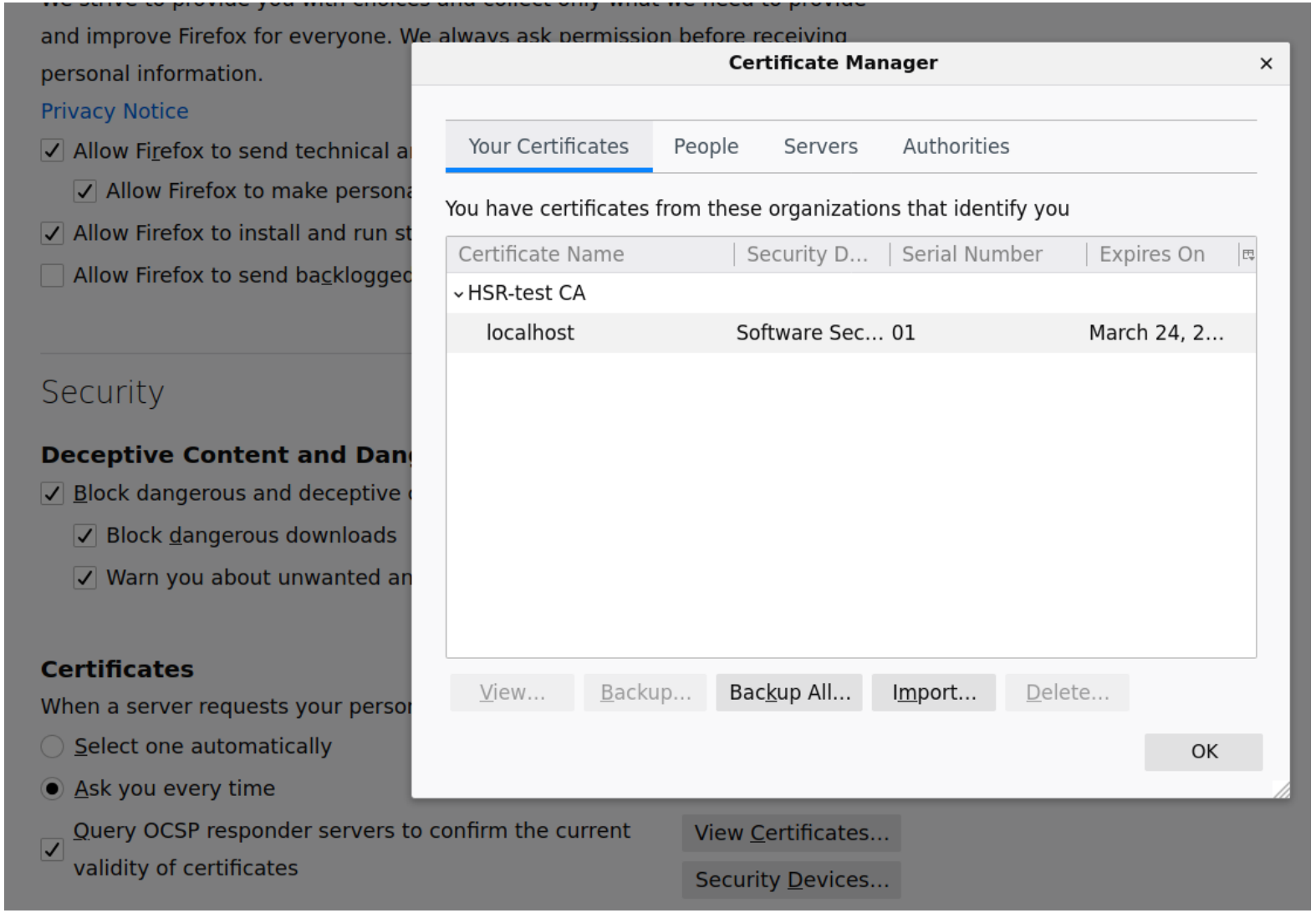
- Create certificate

```
# create signing request
openssl req -new -nodes -keyout user.key -newkey
rsa:4096 -out user.pem
# sign the previous request
openssl x509 -req -days 365 -in user.pem -CA
ca.pem -CAkey ca.key -set_serial 01 -out user.crt
# convert to pkcs12 format
openssl pkcs12 -export -out user.pfx -inkey
user.key -in user.crt -certfile ca.pem
```

- Add nginx security in your local network
  - proxy_ssl_certificate, proxy_ssl_certificate_key



Authentication

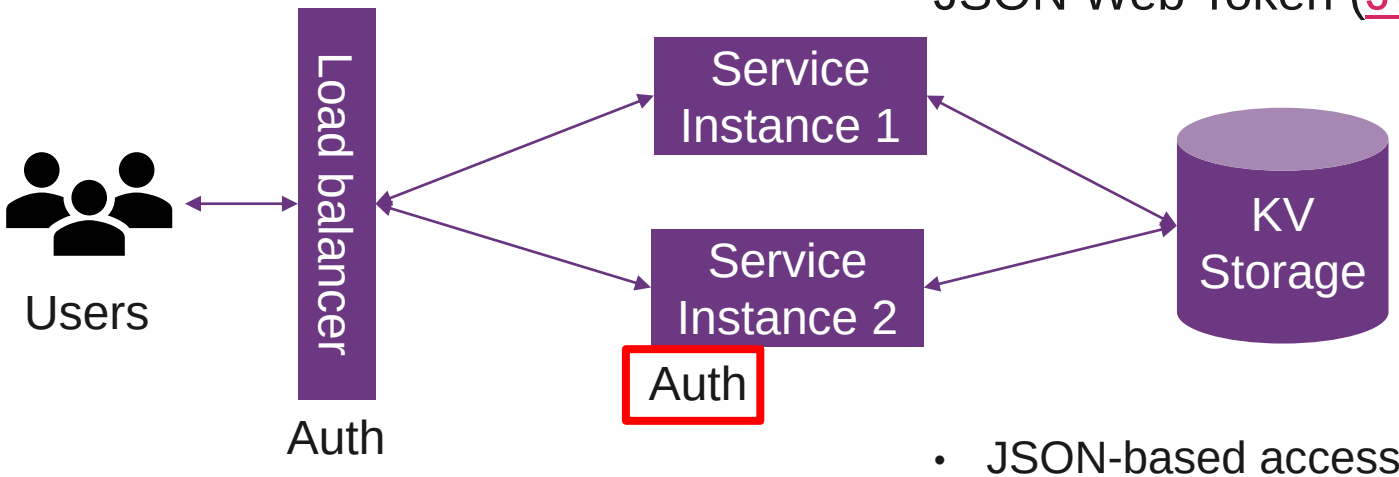


- Add client cert to Firefox
- Used self-signed certificates (for testing)
  - Alternative Let's encrypt

OST

Authentication

- Session-based authentication (stateful)
  - Sticky session



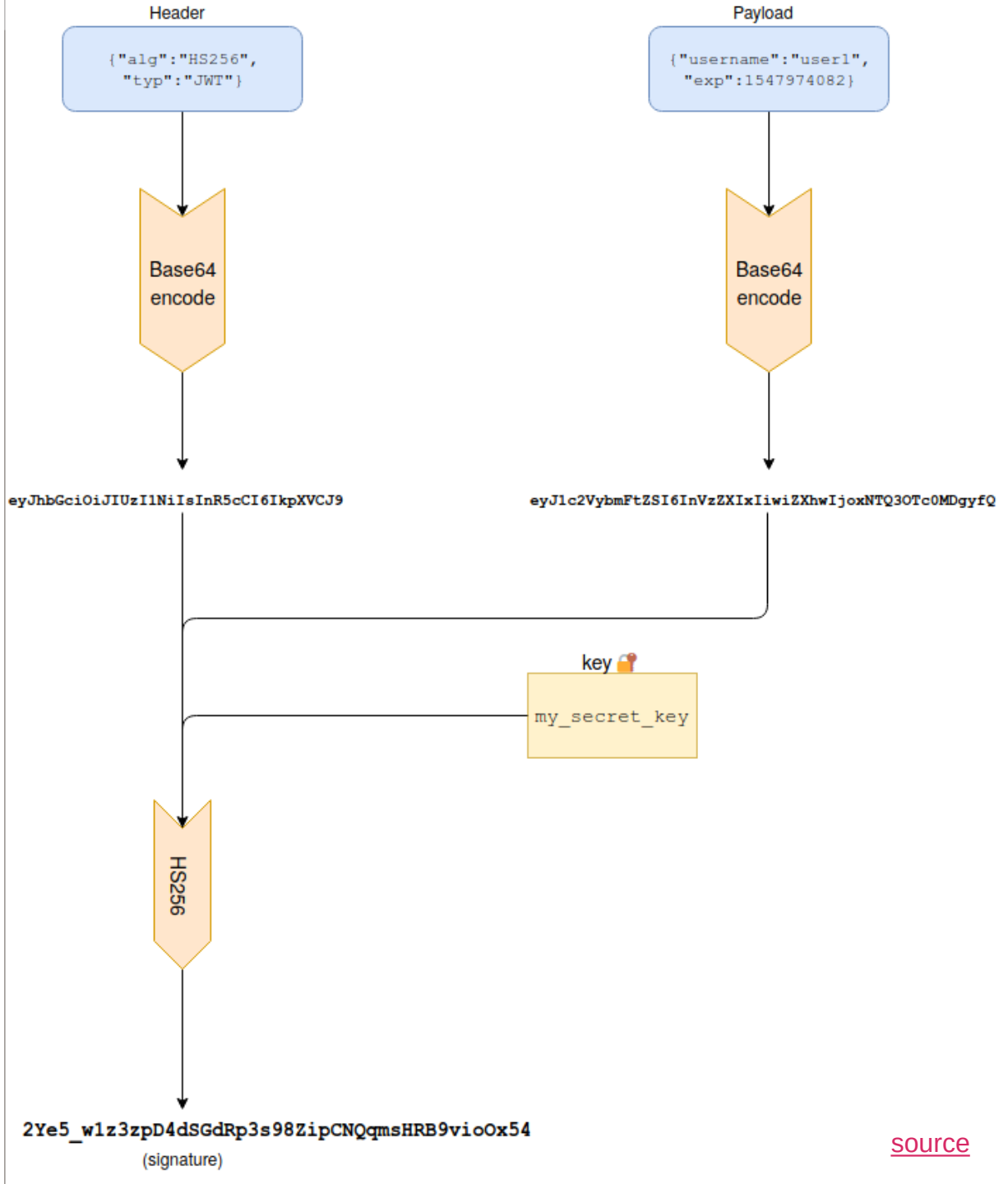
- E.g., spring-boot
  - Simple login app
  - JSESSIONID, Session information, that user was successfully authenticated: memory
  - org.apache.catalina.session.StandardManager based on ConcurrentHashMap
- JSON Web Token (JWT) (stateless)
- JSON-based access tokens (jwt.io)
  - All server instances know a secret token / public key
  - When user logs in, server send back token
  - Client sends: Authorization: Bearer <token>
  - const user_token = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' + base64urlEncoding(signature)

OST

Authentication

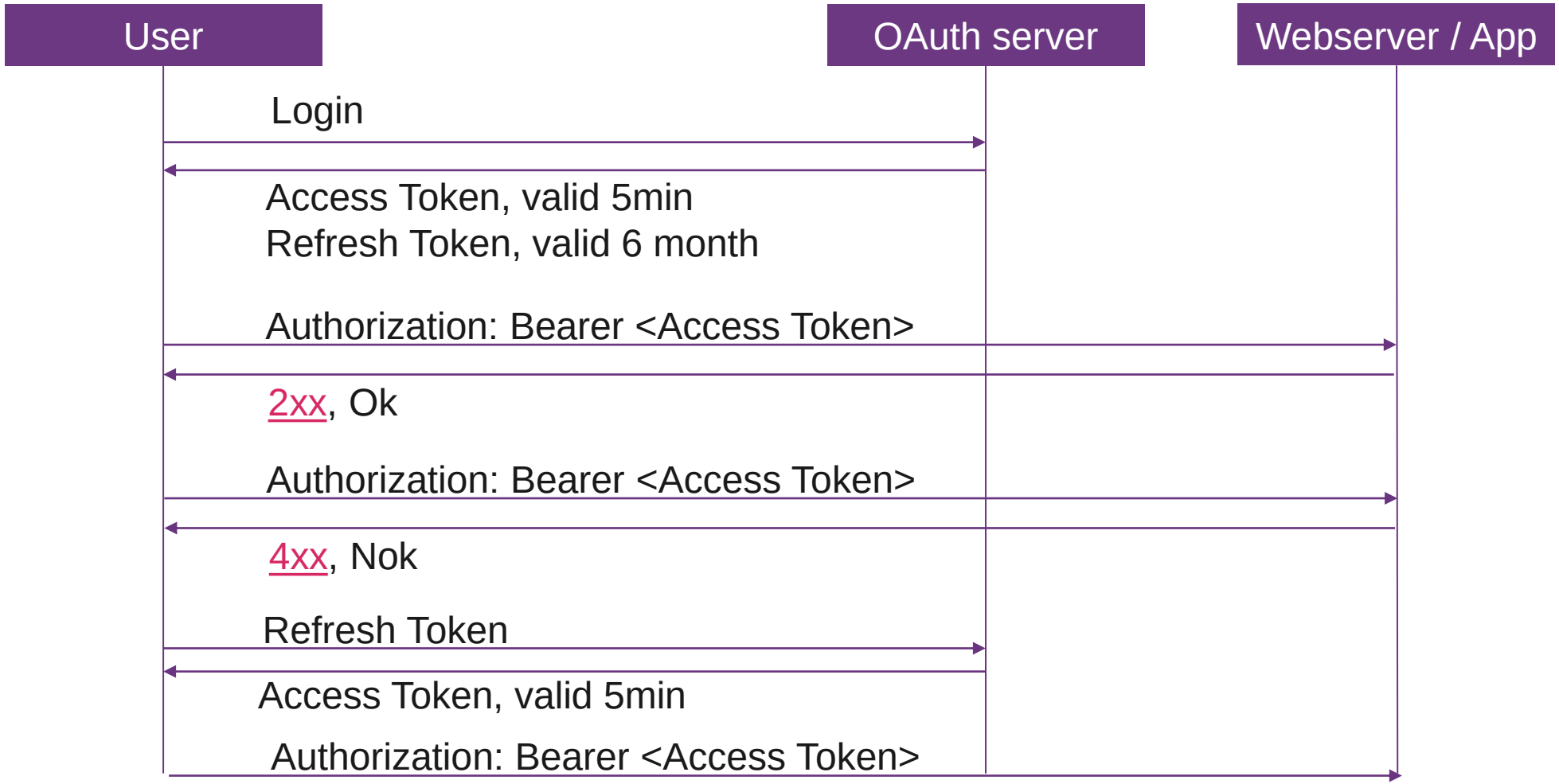
- JSON-based access tokens
  - Header: {"alg" : "HS256"}
  - Payload: {“sub" : "tom", "role" : "admin", “exp" : 1422779638}
- Signature (simple): keyed-hash message
  - ~hash(base64(header)+base64(payload) + secret token)
- Client can store user_token in
  - localStorage.setItem(“token", userToken);
- Example in golang with JWT
  - Tutorial: here and here

- OAuth - protocol for authorization 3rd party integration
  - Grant access on other websites without giving them the passwords



source

Access Token / Refresh Token



OST

Access Token / Refresh Token

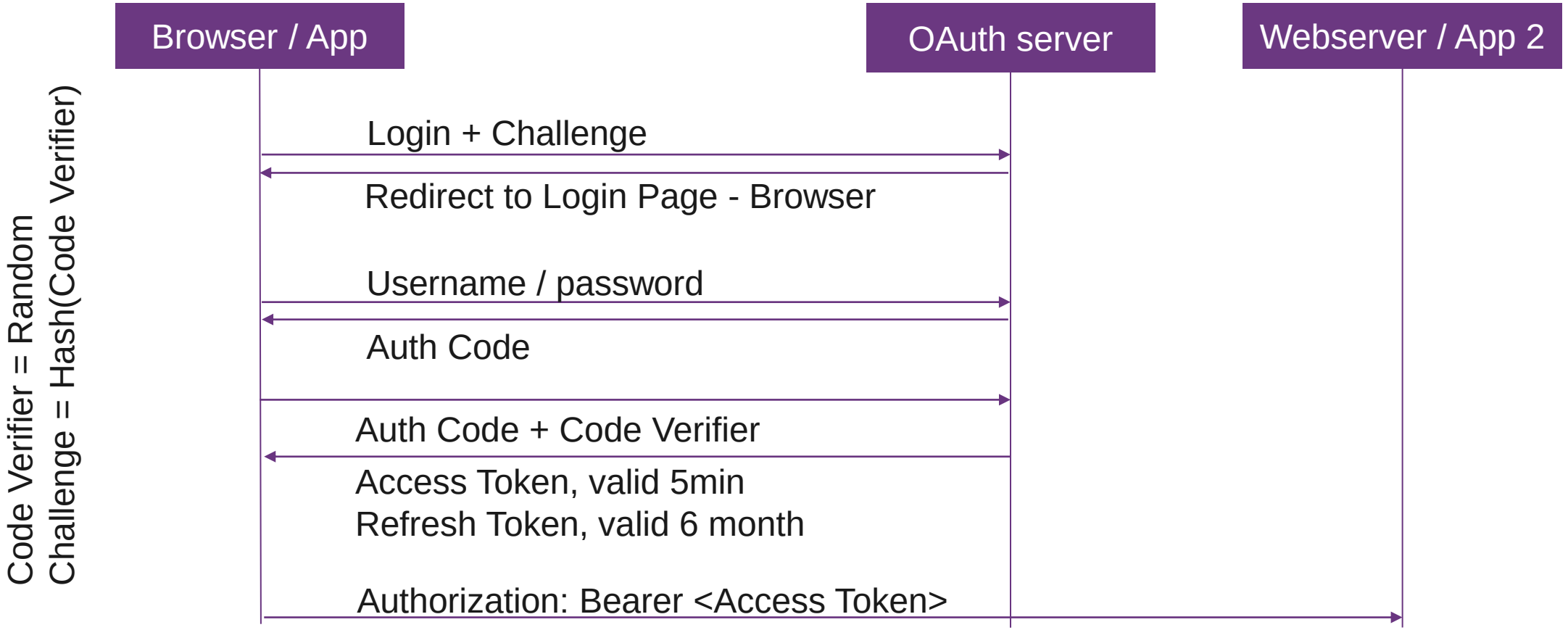
- Access Token only short lifetime, e.g., 10min.
  - If public key / secret is known, the content in the token can be trusted, e.g., in the serivce
  - Can have userId, role, etc.
    - No need to query DB for those information, e.g.:

```
type TokenClaims struct {
        MailFrom string `json:"mail_from,omitempty"`
        MailTo   string `json:"mail_to,omitempty"`
        jwt.Claims
}
```

- Refresh Token longer lifetime, e.g., 6 month
  - A refresh token is used to get a new access token
  - IAM / Auth server creates access tokens

- Only access token, with long lifetime
  - If a user credential is revoked – how to inform every service?
- Only refresh token
  - Tightly coupled Service/Auth, every request to Service, Auth needs to be involved for every access
- Access + Refresh token
  - If a user credential is revoked, user has max. 10min more to access service
  - Auth only involved if access token is expired
- Authorization Code Flow with Proof Key for Code Exchange (PKCE)

OAuth PKCE Flow



Example Auth Server

- Fastauth
  - Simple authentication server that can also be used for local development
  - Start with (also starts LDAP)
    - go build
    - ./fastauth -dev test -rs256 false -ed256 false
    - Login: http://localhost:8080/oauth/authorize?response_type=code
  - PKCE
    - Login:
    - http://localhost:8080/oauth/authorize?response_type=code&code_challenge=fNcHfbUCOvMuzmkBK7c2MR_8TK_Iq6tHDXTJL6qcAco&code_challenge_method=S256
- Copy code from browser
  - curl -X POST -H 'Accept: application/json' --data 'grant_type=authorization_code' --data "code=${CODE}" --data 'code_verifier=HalloDasIstEinTest123456789012345678901234567890' http://localhost:8080/oauth/token
- Now you have the access token
- Example Client with jwt.io
  - example-hello –hs256 test

OST

Authorization Challenge Task

- Minimal Requirement
  - Check a JWT token, token can be generated elsewhere – e.g., jwt.io
  - Can be checked in Service, in LB (traefik only for enterprise version, nginx with plus)

```
[authSources.jwtSource.jwt]
signingSecret = "super-secret"
```

  - Validate in the service

```
func jwtAuth(next func(w http.ResponseWriter, r *http.Request, claims *TokenClaims))
func(http.ResponseWriter, *http.Request) {
        return func(w http.ResponseWriter, r *http.Request) {
                authHeader := r.Header.Get("Authorization")
                if authHeader == "" {
                        writeErr(…)
                        return
                }
                bearerToken := strings.Split(authHeader, " ")
                if len(bearerToken) != 2 {
                        writeErr(…)
                        return
                }
                tok, err := jwt.ParseSigned(bearerToken[1])
                if err != nil {
                        writeErr(…)
                        return
                }
                claims := &TokenClaims{}
                if tok.Headers[0].Algorithm == "HS256" {
                        err = tok.Claims(jwtKey, claims)
                } else {
                        writeErr(…)
                        return
                }
                if err != nil {
                        writeErr(…)
                        return
                }
                if claims.Expiry != nil && !claims.Expiry.Time().After(time.Now()) {
                        writeErr(…)
                        return
                }
                next(w, r, claims)
        }
}
```


OST