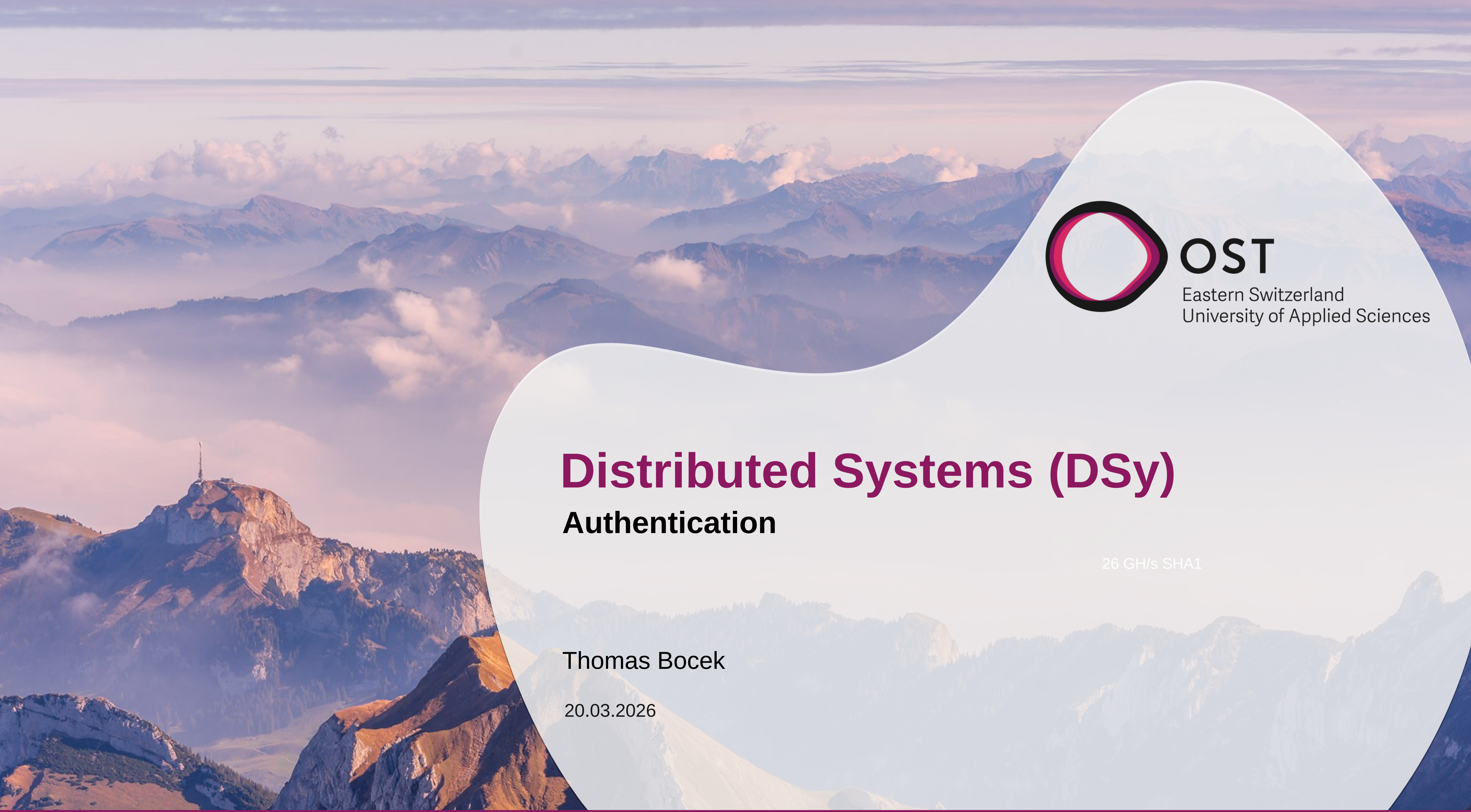


Eastern Switzerland University of Applied Sciences

Distributed Systems (DSy)

Authentication

26 GH/s SHA1

Thomas Bocek

20.03.2026

Learning Goals

- Lecture 6
  - What authentication mechanisms exists for web applications?
  - How can stateless authentication be achieved?

Information security – Key concepts / Access control

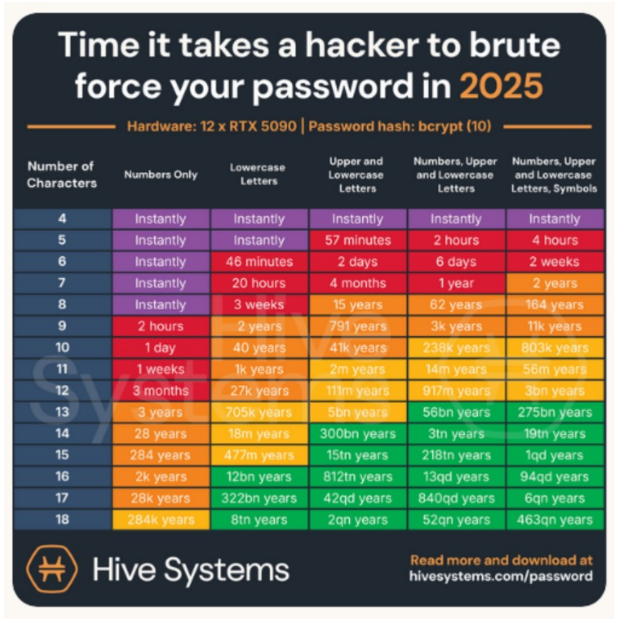
- Confidentiality
  - Protects against eavesdroppers
- Integrity
  - Protection against data modification
- Availability
  - Data needs to be available when needed
- Non-repudiation
  - Neither the sender nor the receiver can deny that a communication has taken place

- Identification
  - E.g. with a username “alice”, claiming to be Alice
- Authentication
  - Verifying a claim of identity. E.g., Alice shows passport, authentication types:
    - Something you know: things such as a PIN, a password
    - Something you have: a key, a swipe card
    - Something you are: biometrics: fingerprint
- Authorization
  - What resources an authenticated user is permitted to access

Authentication

- Authentication
  - Single-factor authentication, e.g. password
  - Multi-factor authentication / 2FA, e.g. password and software token, SMS unsecure
- Password rules - Don’t use: [list]
  - 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 [link] with hashcat - report
  - Hashtype: WPA/WPA2: 1190.5 kH/s
  - SOTA: Argon2id, scrypt, bcrypt, PBKDF2 (salt)



Time it takes a hacker to brute force your password in 2025

Hardware: 12 x RTX 5090 | Password hash: bcrypt (10)

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	57 minutes	2 hours	4 hours
6	Instantly	46 minutes	2 days	6 days	2 weeks
7	Instantly	20 hours	4 months	1 year	2 years
8	Instantly	3 weeks	15 years	62 years	164 years
9	2 hours	2 years	791 years	3k years	11k years
10	1 day	40 years	41k years	238k years	803k years
11	1 weeks	1k years	2m years	14m years	56m years
12	3 months	27k years	111m years	917m years	3bn years
13	3 years	705k years	5bn years	56bn years	275bn years
14	28 years	18m years	300bn years	3tn years	19tn years
15	284 years	477m years	15tn years	218tn years	1qd years
16	2k years	12bn years	812tn years	13qd years	94qd years
17	28k years	322bn years	42qd years	840qd years	6qn years
18	284k years	8tn years	2qn years	52qn years	463qn years

Hive Systems

Read more and download at hivesystems.com/password

https://www.hivesystems.com/password

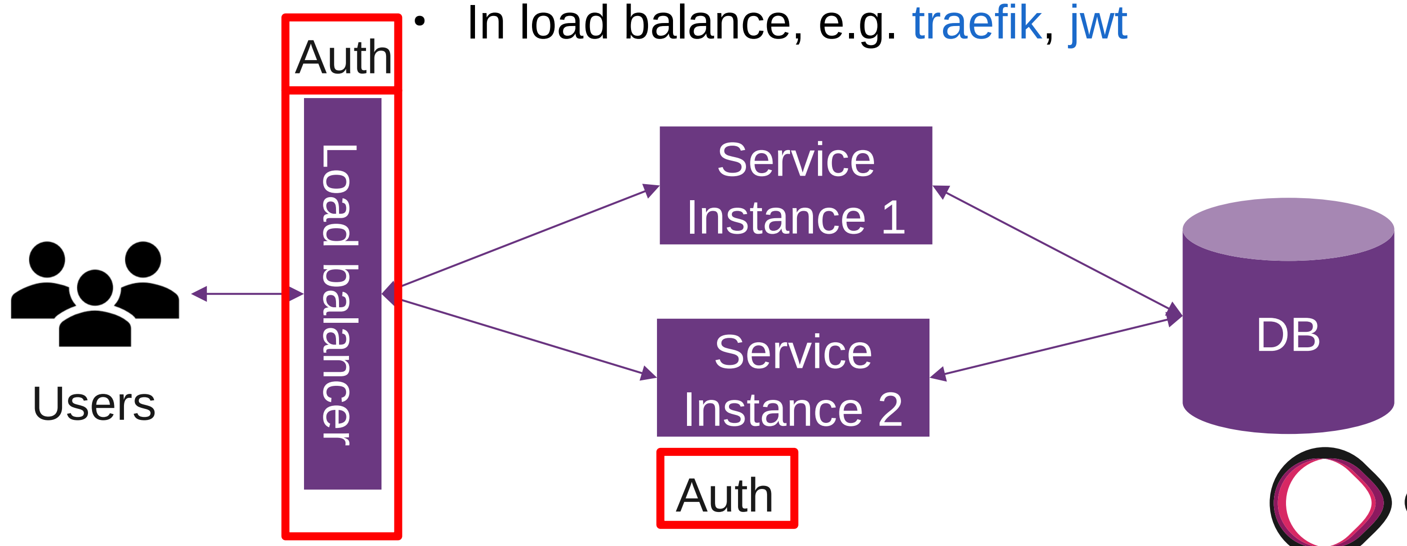
https://www.security.org/how-secure-is-my-password/

Authentication

- Software token: TOTP (Time-based One-time Password)
  - Often used as 2nd factor, e.g., in Google Authenticator, Aegis
  - Based on keyed-hash message authentication code
  - ~ hash(key + message)
  - K=shared secret
  - T=Current Unix time / 30sec
  - TOTP(K, T) = Truncate(HMAC-SHA-256(K,T))

otpauth://totp/Web?secret=password123&issuer=OST

- Where should auth happen?
  - In service, e.g., your HTTP server
  - In load balance, e.g. traefik, jwt



Authentication / Basic Auth

- 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"

Authentication

- Digest Auth
  - 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

Authentication

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

```
client_auth {
          mode require_and_verify
          trusted_ca_cert_file /certs/ca.pem
     }

transport http {
          tls
          tls_client_auth /certs/client.crt /certs/client.key
          tls_trusted_ca_certs /certs/ca.pem
          }
```

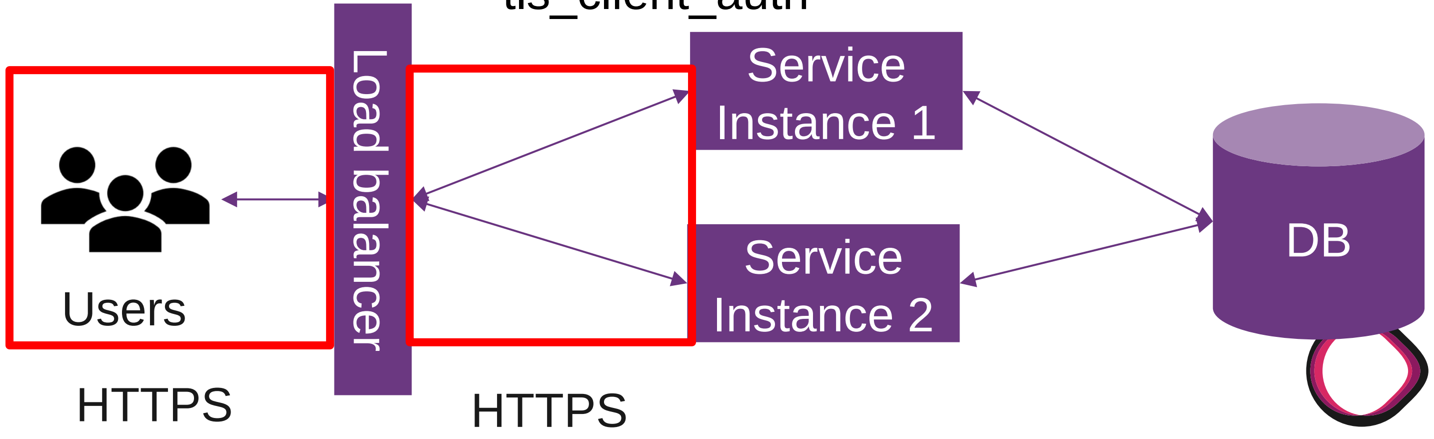
- Create CA / server cert

```
#generate CA certificate and private key
Openssl ...
```

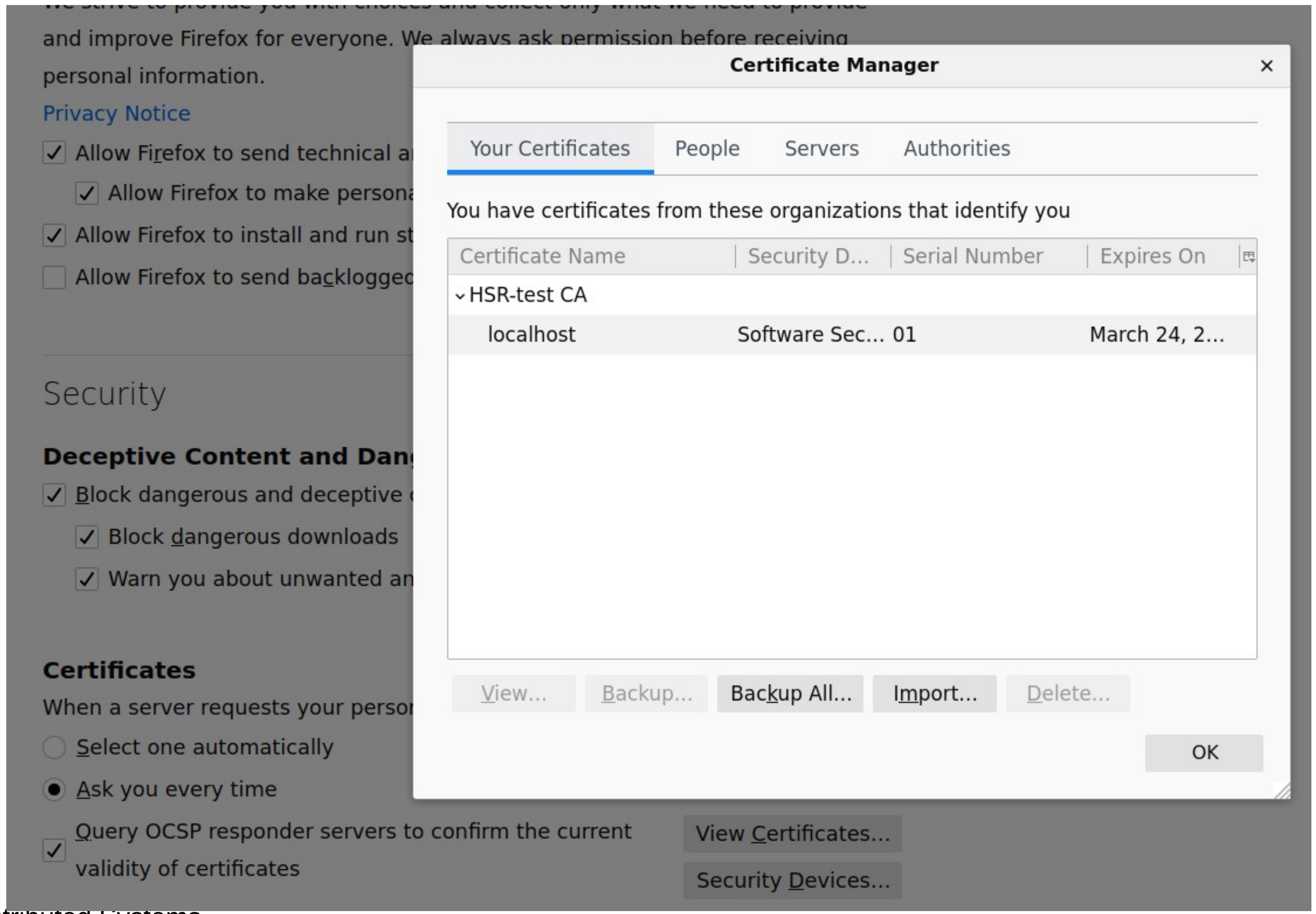
- Create client certificate

```
# create signing request
Openssl ...
```

- Add caddy security in your local network
  - tls_client_auth



Authentication



and improve Firefox for everyone. We always ask permission before receiving personal information.

Privacy Notice

- ✓ Allow Firefox to send technical a
  - ✓ Allow Firefox to make persona
- ✓ Allow Firefox to install and run st
- Allow Firefox to send backlogged

Security

Deceptive Content and Dan

- ✓ Block dangerous and deceptive
  - ✓ Block dangerous downloads
  - ✓ Warn you about unwanted an

Certificates

When a server requests your perso

- Select one automatically
- ● Ask you every time
- ✓ Query OCSP responder servers to confirm the current validity of certificates

View Certificates... | Security Devices...

Certificate Manager

Your Certificates | People | Servers | Authorities

You have certificates from these organizations that identify you

Certificate Name	Security D...	Serial Number	Expires On
HSR-test CA			
localhost	Software Sec...	01	March 24, 2...

View... | Backup... | Backup All... | Import... | Delete... | OK

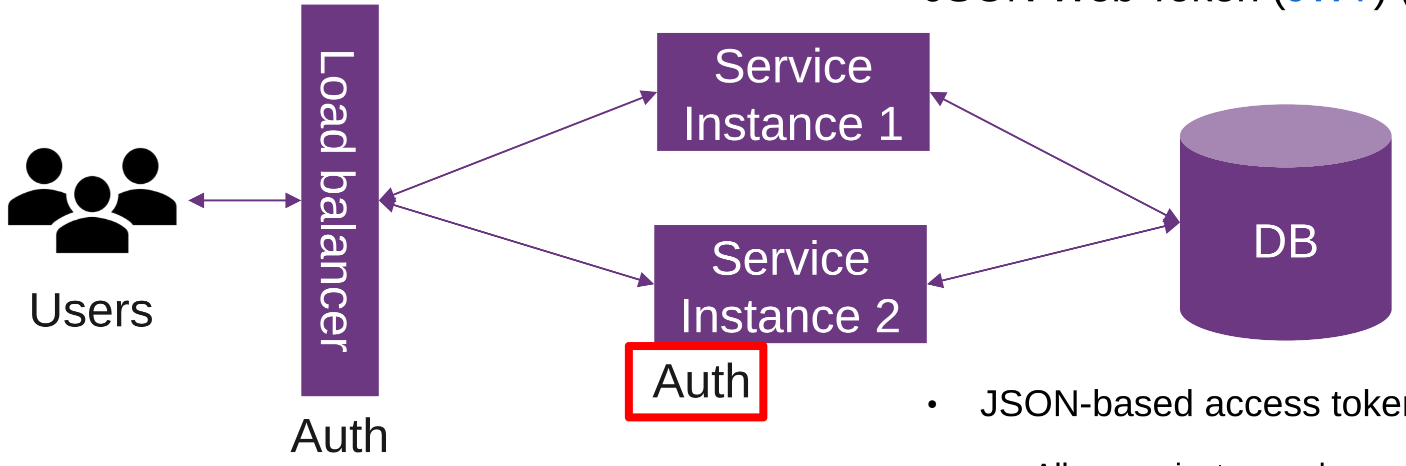
- Add client cert to Firefox
- Used self-signed certificates (for testing)
  - Alternative Let’s encrypt

Lets Encrypt

- Free, automated, open Certificate Authority
- Proves domain control via challenges
- HTTP-Challenge verification
  - Server must be publicly reachable
  - Token placed at /.well-known/acme-challenge/
  - Let's Encrypt verifies token via HTTP request
- DNS-Challenge verification
  - Server does not need to be publicly reachable
  - Enables wildcard certificates
  - Token placed as TXT record in DNS
  - Let's Encrypt verifies the DNS entry
- Certificates valid for 90 days (currently)
  - May 2026: opt-in 45-day certificates
  - Feb 2027: default 64-day certificates
  - Feb 2028: default 45-day certificates
  - Optional: 6-day short-lived certificates (since 2026)
- Integrated in modern web servers (e.g., Caddy)`

Authentication

- Session-based authentication (stateful)
  - Sticky session



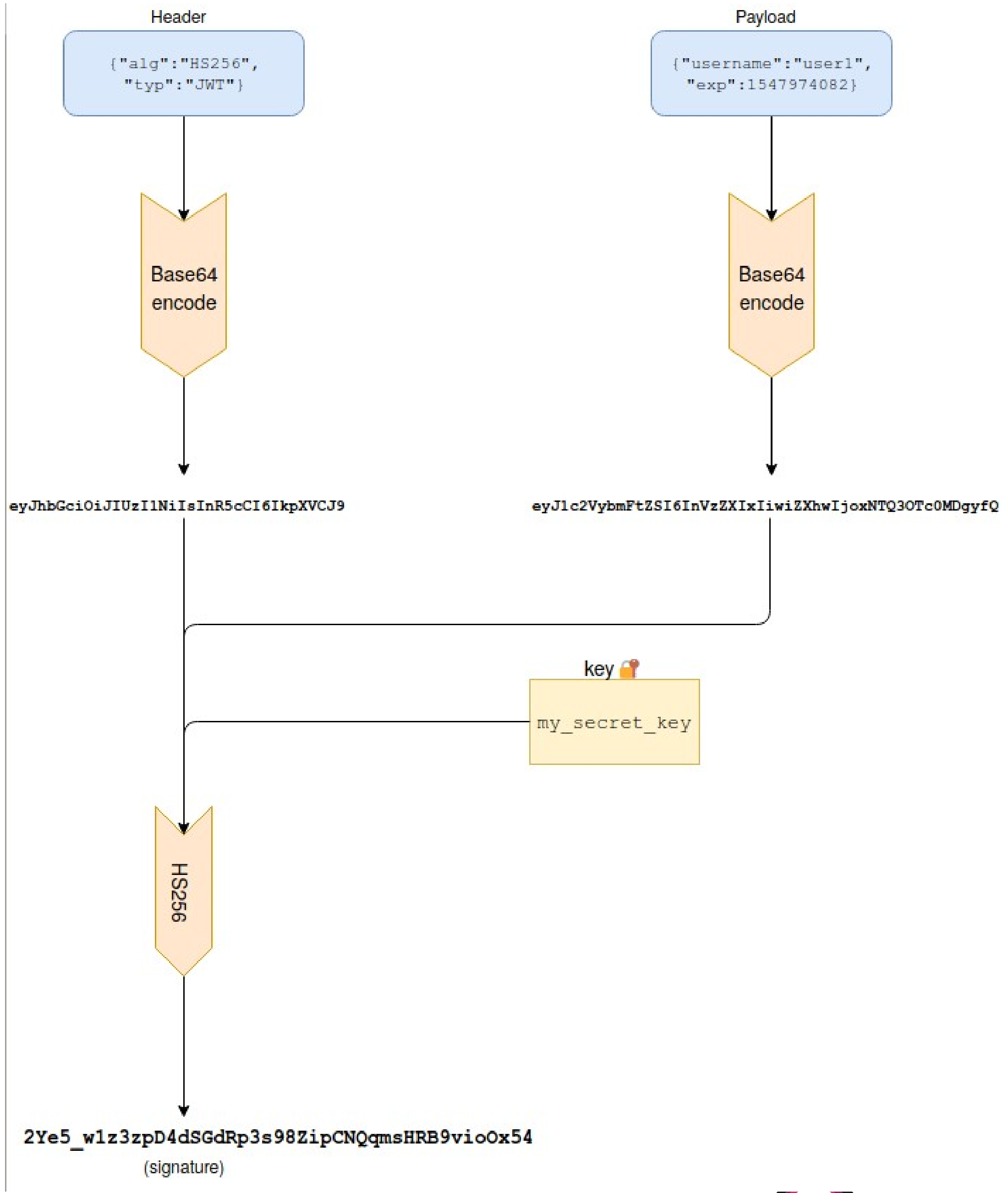
- E.g., spring-boot
  - Simple login app
  - JSESSIONID, Session information, that user was successfully authenticated: memory

- org.apache.catalina.session.StandardManager based on ConcurrentHashMap

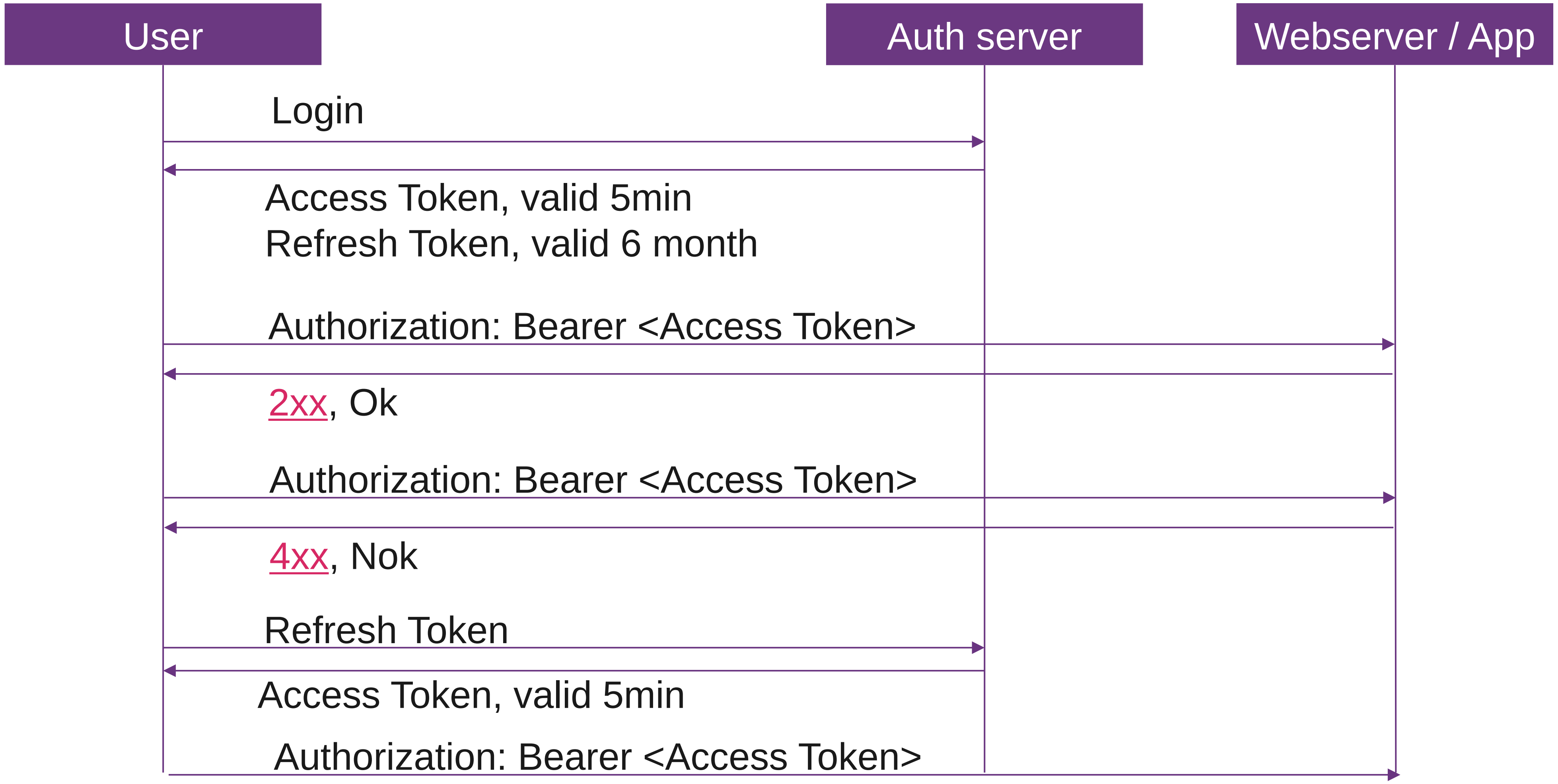
- JSON Web Token (JWT) (stateless)
  - JSON-based access tokens (jwt.io)
    - All server instances know a secret token / public key
    - When user logs in, server send back token
    - Client sends: Authorization: Bearer <token>
    - const user_token = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' + base64urlEncoding(signature)

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)
- Token = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' + base64urlEncoding(signature)
- Client can store token in localStorage.setItem(“token", accessToken);
- Example in golang with JWT, tutorial: here



Access Token / Refresh Token


User
Auth server
Webserver / App
Login
Access Token, valid 5min
Refresh Token, valid 6 month
Authorization: Bearer <Access Token>
2xx, Ok
Authorization: Bearer <Access Token>
4xx, Nok
Refresh Token
Access Token, valid 5min
Authorization: Bearer <Access Token>

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
}
```

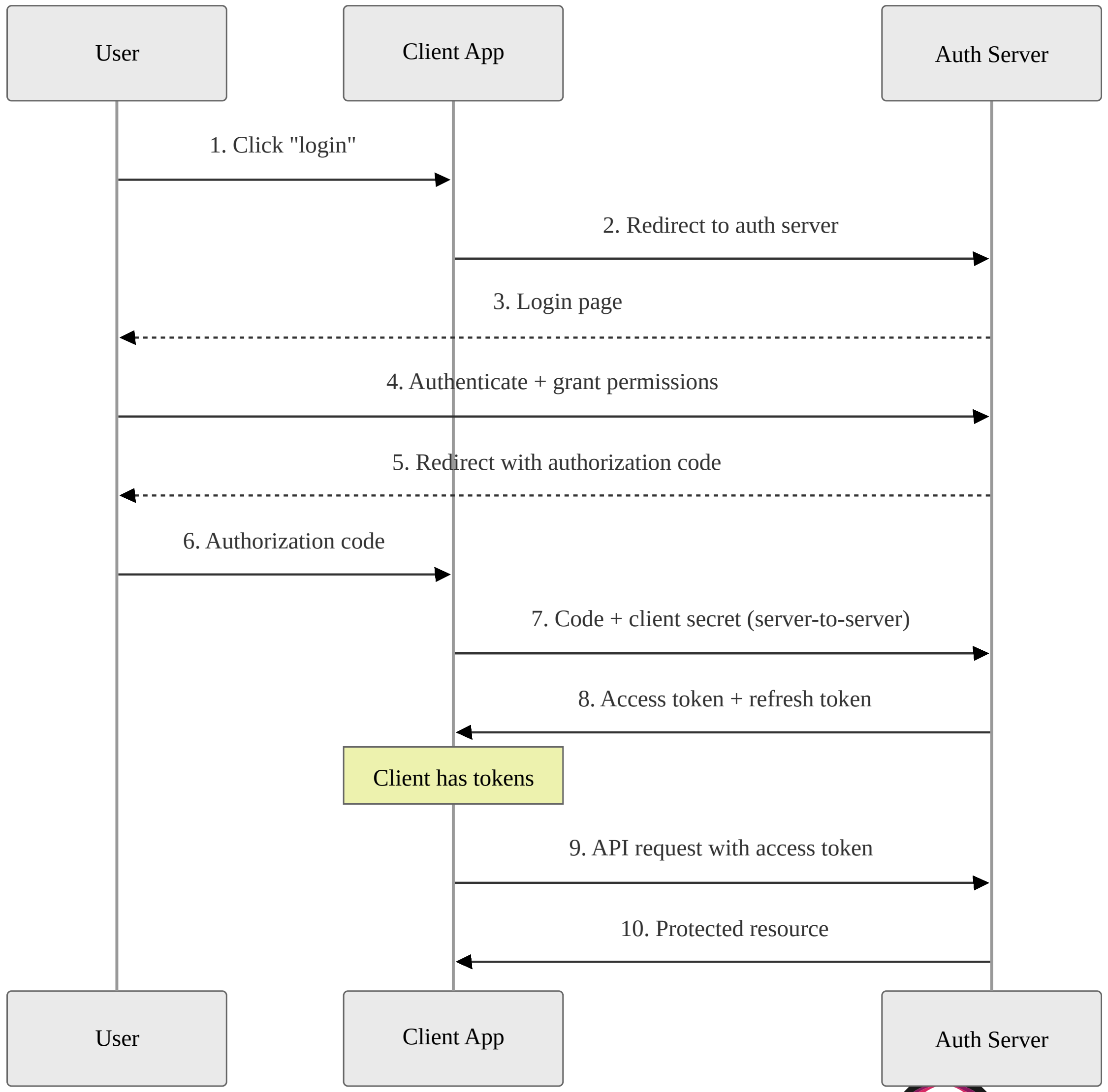
- Refresh Token longer lifetime, e.g., 6 month
  - A refresh token is used to get a new access token
  - IAM / Auth server creates access tokens

- 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

OAuth

- OAuth for authorization 3rd party integration
  - Grant access without giving away passwords
  - Flows (Authorization Code Grant, PKCE, …)
- Authorization code grant
  - User redirected to authorization server, user authenticates and grants permissions, authorization code returned to app
  - App exchanges code + client secret for tokens
- PKCE: variant for clients that cannot store a client secret securely (mobile, SPA)
- Tokens issued are typically JWTs (access + refresh)

Authorization code grant


User
Client App
Auth Server
1. Click "login"
2. Redirect to auth server
3. Login page
4. Authenticate + grant permissions
5. Redirect with authorization code
6. Authorization code
7. Code + client secret (server-to-server)
8. Access token + refresh token
Client has tokens
9. API request with access token
10. Protected resource
User
Client App
Auth Server