Application Security (apsi)

Lecture at FHNW

Lecture 13 2021

Arno Wagner, Michael Schläpfer, Rolf Wagner

<arno@wagner.name>, <{michael.schlaepfer,rolf.wagner}@fort-it.ch>

Agenda

- **Identification, Authentication, Authorization, Access**
- Recap: Public Key Cryptography
- JSON Object Signing and Encryption (JOSE)
- Authorization with OAuth
- Identity Federation (OpenID Connect, SAML)

Authentication

Authentication vs. Identification

- Identification: Statement of identity
- Authentication: Act of verifying or confirming an identity

Authentication usually uses cryptographic methods

- Public-key signatures, encrypted transmission of a secret

Authentication has phases

1. Initial establishment
2. Maintaining it during a session
    - By knowledge of the (secret) session key: SSL/TLS
    - By possession of a session cookie (to be transferred only over a secure channel)
    - By echoing a secret the other side included in the last transmission
3. Removal of the authentication

Authorization

Authorization is the process of assigning access permissions

- Usually, some form of authenticated identity serves as basis
- A technical state (established connection, knows a secret, access-attempt at specific time, etc.) can also be used. An identity is optional!
- Assigning default rights is also authorization

Authorization is based on policies

- Example: Same-Origin Policy
- Example: HTTPS-only

Access to a Resource

If the access policy requires a specific user, the following steps are typically performed:

1) User is identified
2) Identity is authenticated
3) User identity is checked against the policy
4) Access is granted or denied

Example: Login with user name and password, then access a file

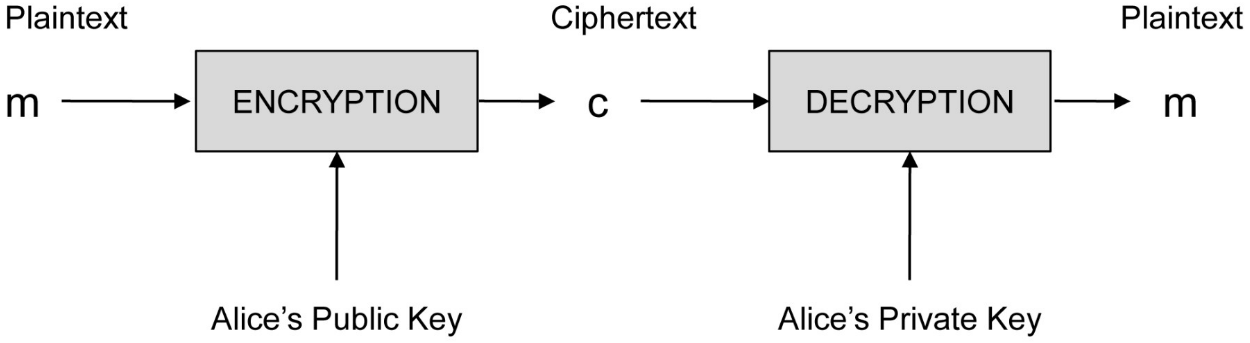
- User name refers to the identity
- Password provided is the authenticator
- Authentication is maintained by login-session (telnet: weak, ssh: strong)
- File to be accessed has permissions and Owner/Group/other
  File access policy and identity of user is checked to determine access

Agenda

- Identification, Authentication, Authorization, Access
- **Recap: Public Key Cryptography**
- JSON Object Signing and Encryption (JOSE)
- Authorization with OAuth
- Identity Federation (OpenID Connect, SAML)

Recap: Public Key Cryptography

- Private/public key pairs based on mathematical problems
- Main use cases:
  - Encryption
  - Digital signatures
- Public Key Infrastructure for certifying ownership of key pairs
- Alternative: Web of trust (e.g., PGP)



Agenda

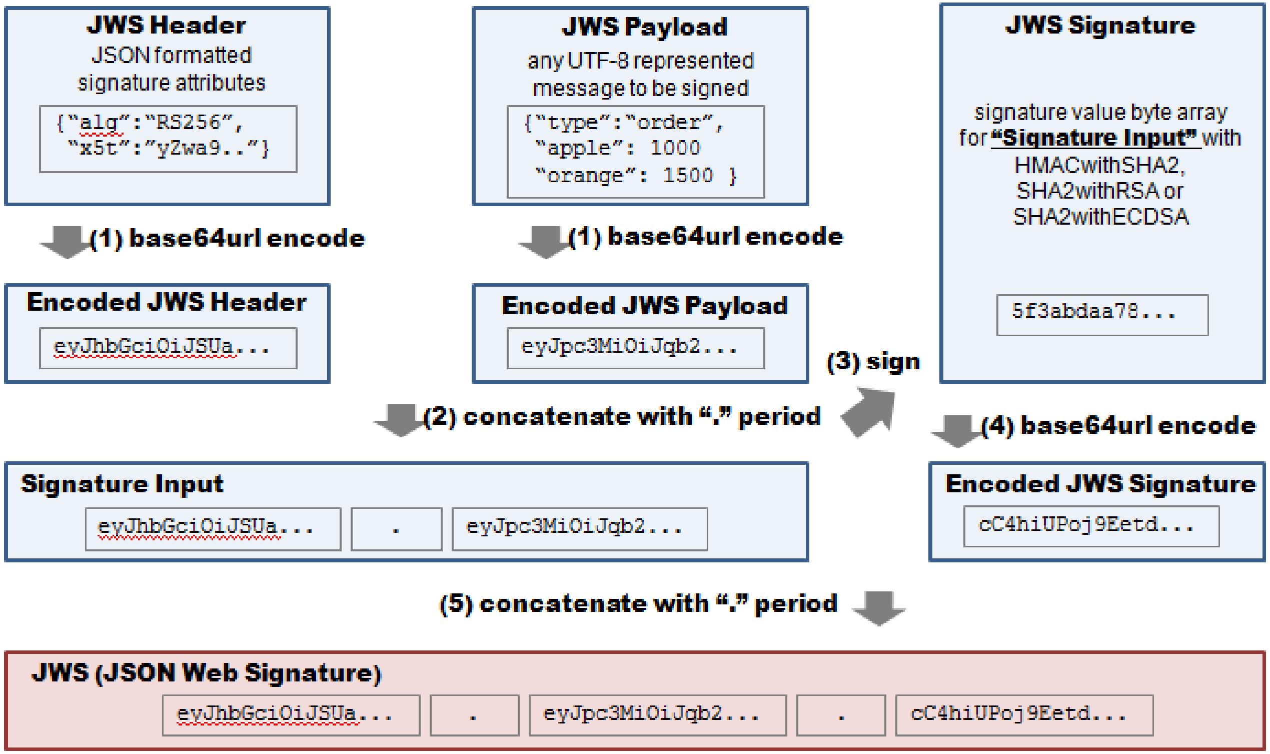
- Identification, Authentication, Authorization, Access
- Recap: Public Key Cryptography
- **JSON Object Signing and Encryption (JOSE)**
- Authorization with OAuth
- Identity Federation (OpenID Connect, SAML)

JSON Object Signing and Encryption (JOSE)

- 3 basic JSON (www.json.org, RFC 7159) object formats:
  - Integrity-protected object format: JWS (RFC 7515)
  - Confidentiality-protected object format: JWE (RFC 7516)
  - Format for expressing keys: JWK (RFC 7517)
- Details on algorithms: JWA (RFC 7518)
- Used for:
  - Security Tokens (JWT)
  - OAuth
  - OpenID Connect
- JSON vs. compact serialization (URL-safe representation, RFC 7515)

JSON Web Signatures (JWS)

- Represents content secured with digital signatures or Message Authentication Codes (MACs)



JWS Header

JSON formatted signature attributes

```
{"alg":"RS256",
 "x5t":"yZwa9.."}
```

JWS Payload

any UTF-8 represented message to be signed

```
{"type":"order",
 "apple": 1000
 "orange": 1500 }
```

JWS Signature

signature value byte array for **"Signature Input"** with HMACwithSHA2, SHA2withRSA or SHA2withECDSA

```
5f3abdaa78...
```

(1) base64url encode

(1) base64url encode

Encoded JWS Header

```
eyJhbGciOiJSUa...
```

Encoded JWS Payload

```
eyJpc3MiOiJqb2...
```

(3) sign

(2) concatenate with "." period

(4) base64url encode

Signature Input

```
eyJhbGciOiJSUa...  .  eyJpc3MiOiJqb2...
```

Encoded JWS Signature

```
cC4hiUPoj9Eetd...
```

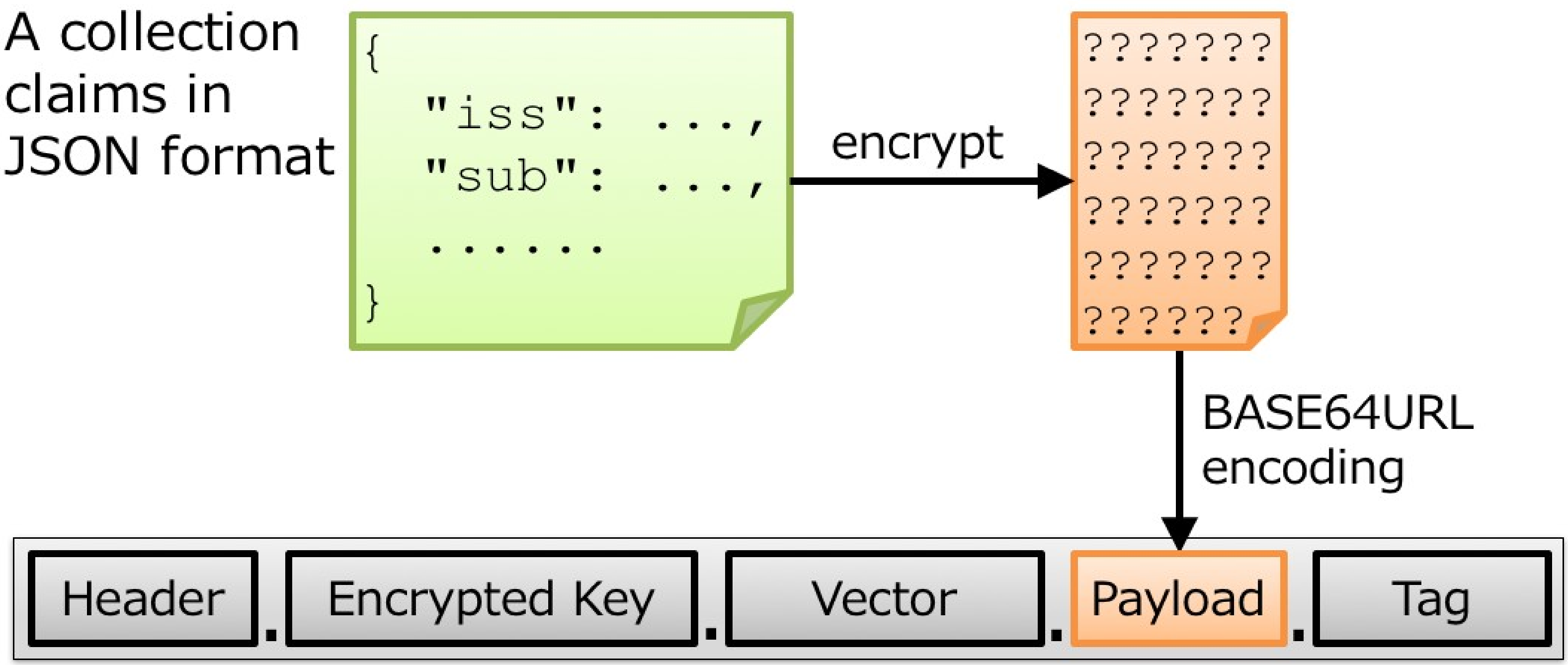
(5) concatenate with "." period

JWS (JSON Web Signature)

```
eyJhbGciOiJSUa...  .  eyJpc3MiOiJqb2...  .  cC4hiUPoj9Eetd...
```

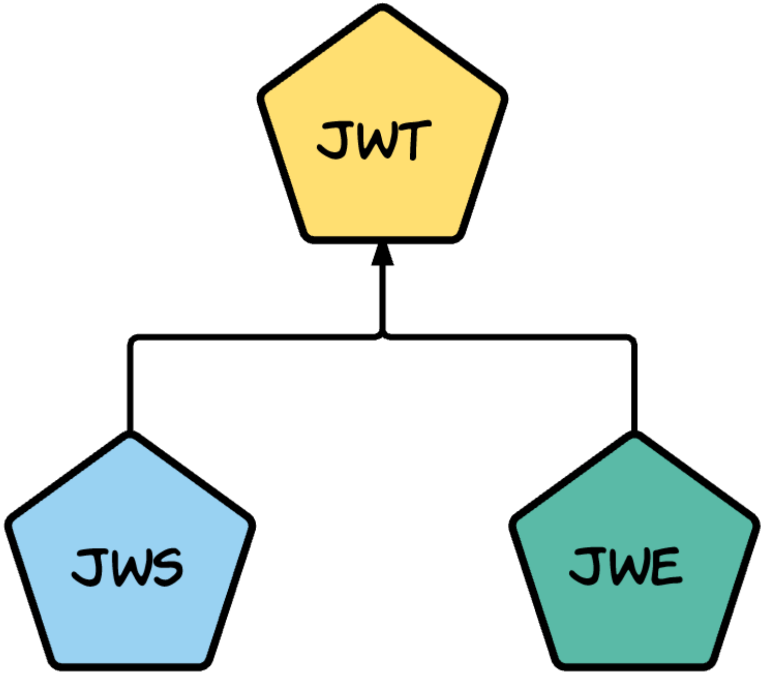
JSON Web Encryption (JWE)

- Represents encrypted content



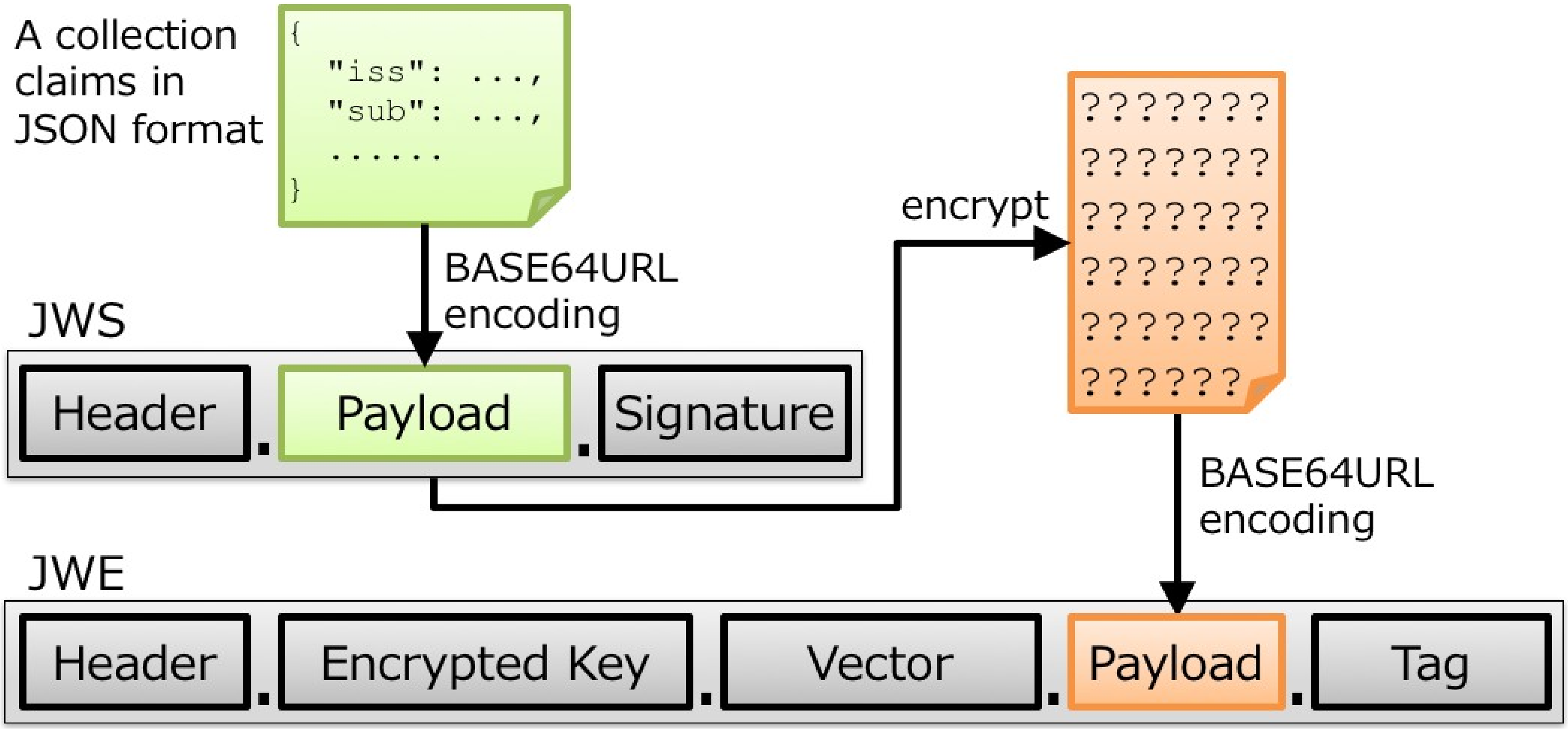
JSON Web Token (JWT)

- RFC 7519
- Represents security tokens meant to be distributed between computer systems
- Compact serialization
- URL-safe representation of claims
- Information about subject as claims
- JWS, JWE or both (nested)



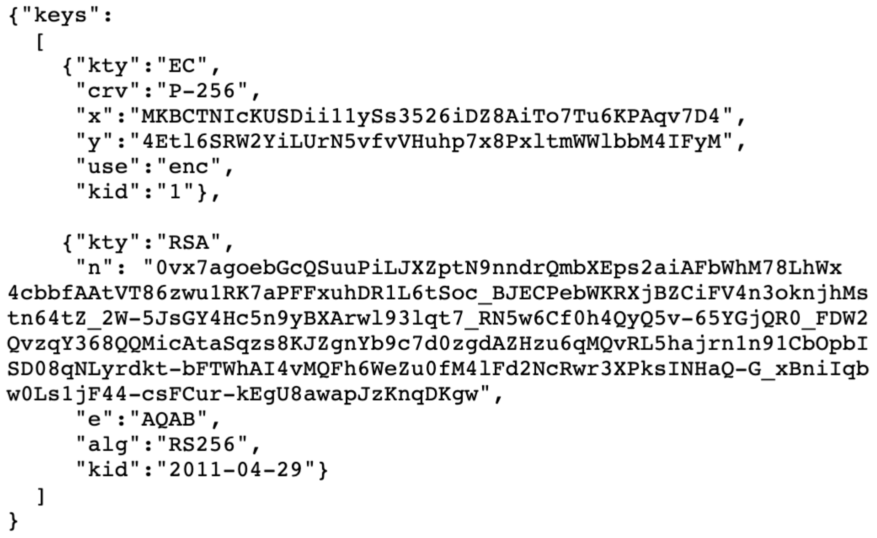
Nested JWT

- First sign, then encrypt!



JOSE Implementation Details

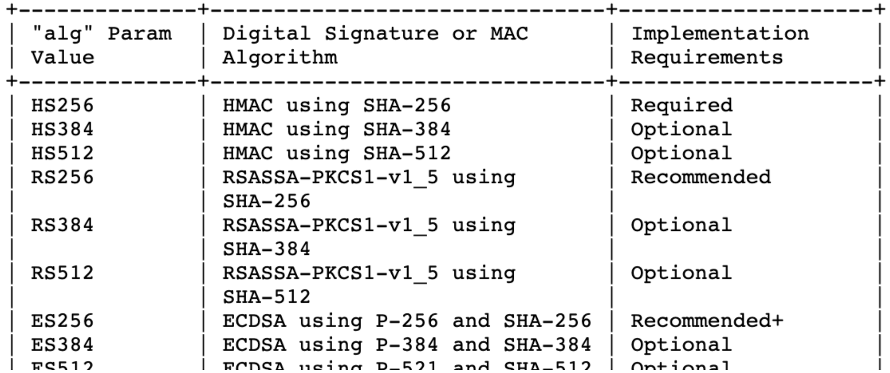
- JSON Web Key (JWK):
  - JSON data structure representing a cryptographic key or a set thereof



```
{"keys":
  [
    {"kty":"EC",
     "crv":"P-256",
     "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
     "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
     "use":"enc",
     "kid":"1"},

    {"kty":"RSA",
     "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
     "e":"AQAB",
     "alg":"RS256",
     "kid":"2011-04-29"}
  ]
}
```

- JSON Web Algorithms (JWA):
  - Cryptographic algorithms and identifiers to be used with JWS, JWE, and JWK
  - Recommendations



"alg" Param Value	Digital Signature or MAC Algorithm	Implementation Requirements
HS256	HMAC using SHA-256	Required
HS384	HMAC using SHA-384	Optional
HS512	HMAC using SHA-512	Optional
RS256	RSASSA-PKCS1-v1_5 using SHA-256	Recommended
RS384	RSASSA-PKCS1-v1_5 using SHA-384	Optional
RS512	RSASSA-PKCS1-v1_5 using SHA-512	Optional
ES256	ECDSA using P-256 and SHA-256	Recommended+
ES384	ECDSA using P-384 and SHA-384	Optional
ES512	ECDSA using P-521 and SHA-512	Optional

Agenda

- Identification, Authentication, Authorization, Access
- Recap: Public Key Cryptography
- JSON Object Signing and Encryption (JOSE)
- **Authorization with OAuth**
- Identity Federation (OpenID Connect, SAML)

OAuth 2.0

Reference: IETF RFC 6749 "The OAuth 2.0 Authorization Framework"

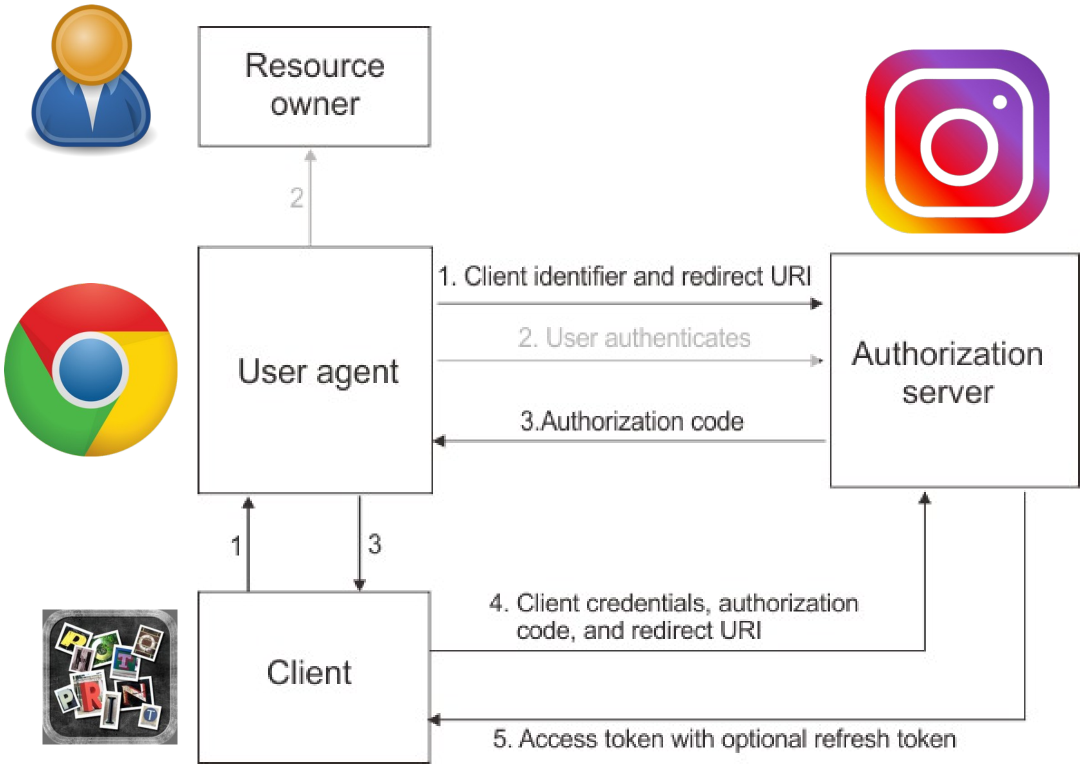
- There is "authorization" in the name…
- Note: Different implementations may not interact

What is it? (simplified)

- Used to allow users to authorize a web-site to give their information to a different site or to an application, without sharing their password
- Also known as "secure delegated access"

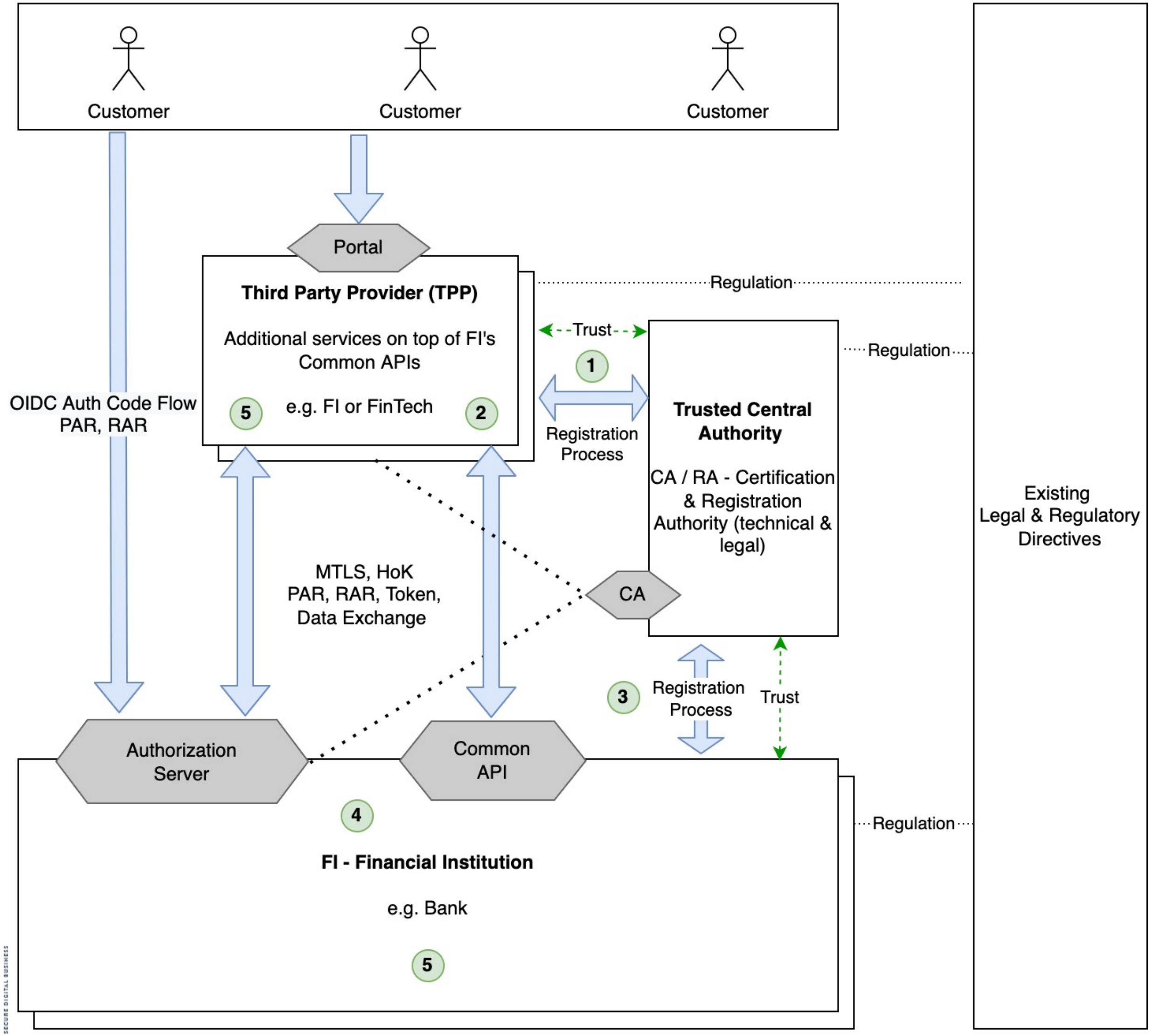
OAuth 2.0 Authorization Code Flow

Use case: User intends to authorize a ***Photoprint App*** to directly access her pictures in ***Instagram***.


Resource owner
2
User agent
1. Client identifier and redirect URI
2. User authenticates
3.Authorization code
Authorization server
1
3
Client
4. Client credentials, authorization code, and redirect URI
5. Access token with optional refresh token

Example OIDF: Financial-grade API (FAPI)

- https://fapi.openid.net/

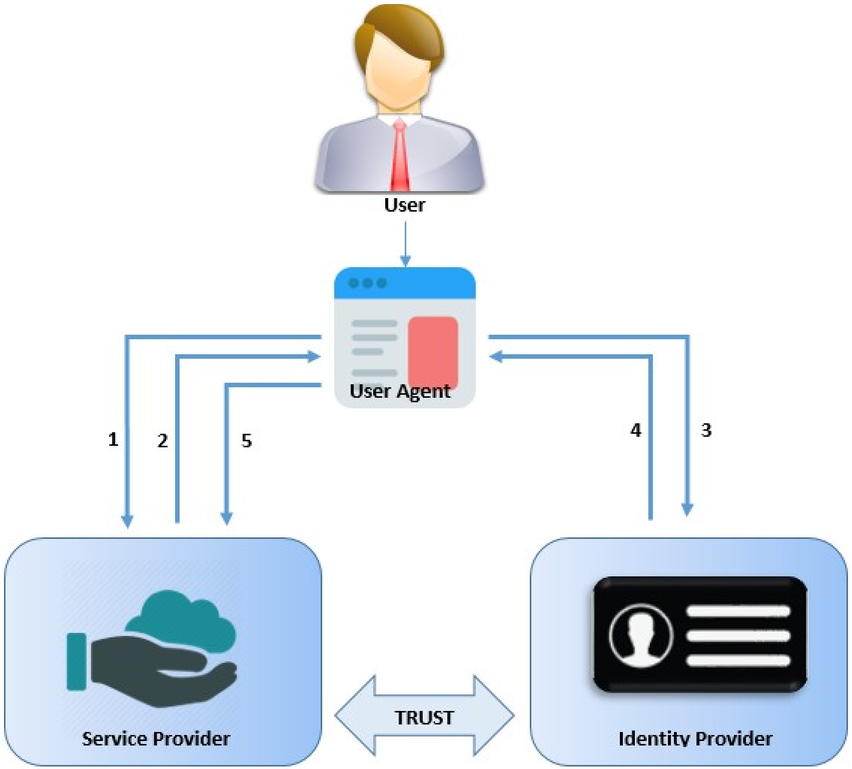


Agenda

- Identification, Authentication, Authorization, Access
- Recap: Public Key Cryptography
- JSON Object Signing and Encryption (JOSE)
- Authorization with OAuth
- **Identity Federation (OpenID Connect, SAML)**

Identity Federation

- Allows Cross-domain Single-sign-on
- Widely deployed: SAML, OpenID Connect



OpenID Connect

OpenID Connect (OIDC) is an authentication layer on top of OAuth 2.0

RESTful HTTP API, using JSON as a data format

Adds a signed Identity Token that can be verified and parsed by the client (The OAuth access token is opaque)

Reference: https://oauth.net/articles/authentication/

Flow is almost like for OAuth, but

- A signed token stating the identity of the user is given to the client
- The identity provider always authenticates the user

Recent OAuth 2.0 Flaws

Really a number of flaws:

- Accepting the identity without verifying it is tied to the OAuth access token
  Note: This misuses OAuth as authentication protocol in addition
- OpenID Connect is used, but the signature of the identity token is not verified
- User Identity is not tied to OAuth exchange, but retrieved locally
  Note: This misuses OAuth as authentication protocol in addition

What happened here?

- Using OAuth and OpenID connect without understanding it
- Testing was only done to verify it works, but security was not tested at all

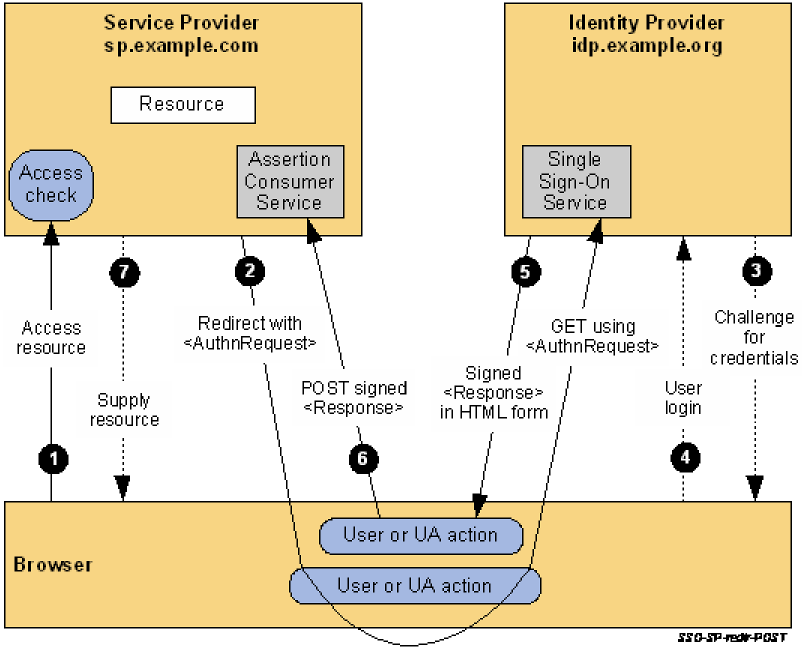
Reference: "Signing Into Billion Mobile Apps Effortlessly With OAuth 2.0", Yang et. al.

SAML 2.0

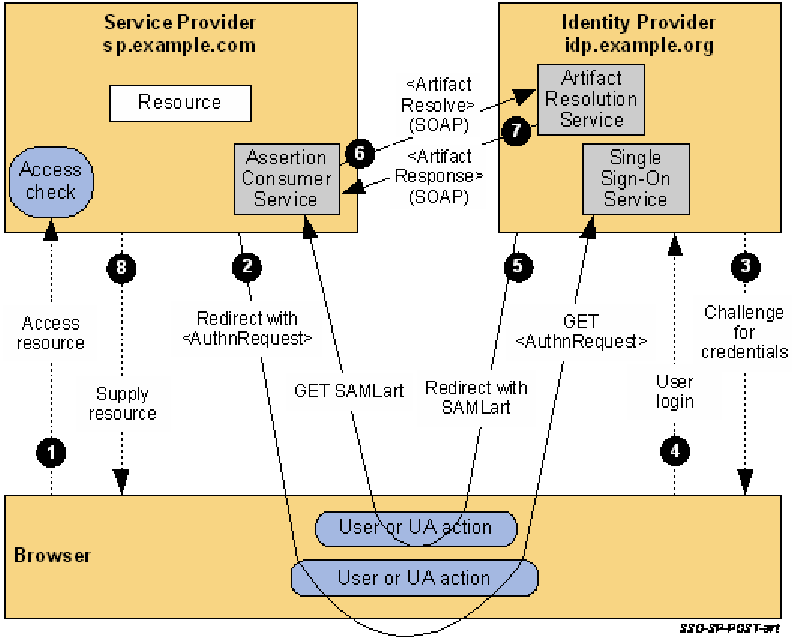
- Security Assertion Markup Language
- Versions before 2.0 where ambiguous / proprietary (MS, SAP, …)
- SAML 2.0 ratified as an OASIS standard in 2005
- Still de facto standard for Identity Federation (e.g., for SaaS)
- Different concepts: profiles, bindings, protocols, assertions
- Core element: Assertion (XML file with information about subject)

SAML 2.0

- Most common: POST-binding and Artifact-binding



Service Provider
sp.example.com
Resource
Access check
Assertion Consumer Service
Identity Provider
idp.example.org
Single Sign-On Service
1
Access resource
7
Supply resource
2
Redirect with <AuthnRequest>
POST signed <Response>
6
5
Signed <Response> in HTML form
GET using <AuthnRequest>
User login
4
3
Challenge for credentials
Browser
User or UA action
User or UA action
SSO-SP-redir-POST



Service Provider
sp.example.com
Resource
Access check
Assertion Consumer Service
<Artifact Resolve> (SOAP)
6
7
<Artifact Response> (SOAP)
Identity Provider
idp.example.org
Artifact Resolution Service
Single Sign-On Service
1
Access resource
8
Supply resource
2
Redirect with <AuthnRequest>
GET SAMLart
Redirect with SAMLart
5
GET <AuthnRequest>
User login
4
3
Challenge for credentials
Browser
User or UA action
User or UA action
SSO-SP-POST-art