Using DPoP to use access tokens securely in your Single Page Applications

Université Libre de Bruxelles Campus du Solbosch, Brussels, Belgium

1 February 2025

Alexander Schwartz, Principal Software Engineer, Keycloak Maintainer @ Red Hat
Takashi Norimatsu, Senior OSS Specialist, Keycloak Maintainer @ Hitachi, Ltd.

Self Introduction

Alexander Schwartz (**ahus1** in GitHub) :

- Keycloak maintainer (since Jun 2023)
- Principal Software Engineer, Red Hat, Germany 
- Key areas: Keycloak high availability, load testing, observability
- Contributing to Keycloak since 2015

Takashi Norimatsu (**tnorimat** in GitHub) :

- Keycloak maintainer (since Oct 2021),
- Technical lead of Keycloak community “OAuth SIG” (since Aug 2020),
- Ph.D. Student, Senior OSS Specialist, Hitachi, Ltd., Japan 
- Certified Information Systems Security Professional (CISSP),
- Key areas: Security protocols, API security
- Contributing to Keycloak since 2017

HITACHI
Inspire the Next

Contents

HITACHI
Inspire the Next

1. SPA in OAuth 2.0
2. Security Risk
3. Countermeasure: DPoP
4. Security Considerations

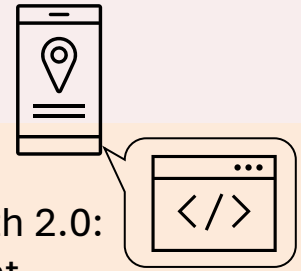
SPA in OAuth 2.0

HITACHI
Inspire the Next

End User

OAuth 2.0:
Resource
Owner

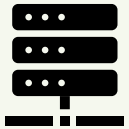
1. Access
Browser



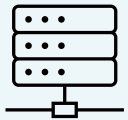
OAuth 2.0:
Client

SPA

Authorization Server



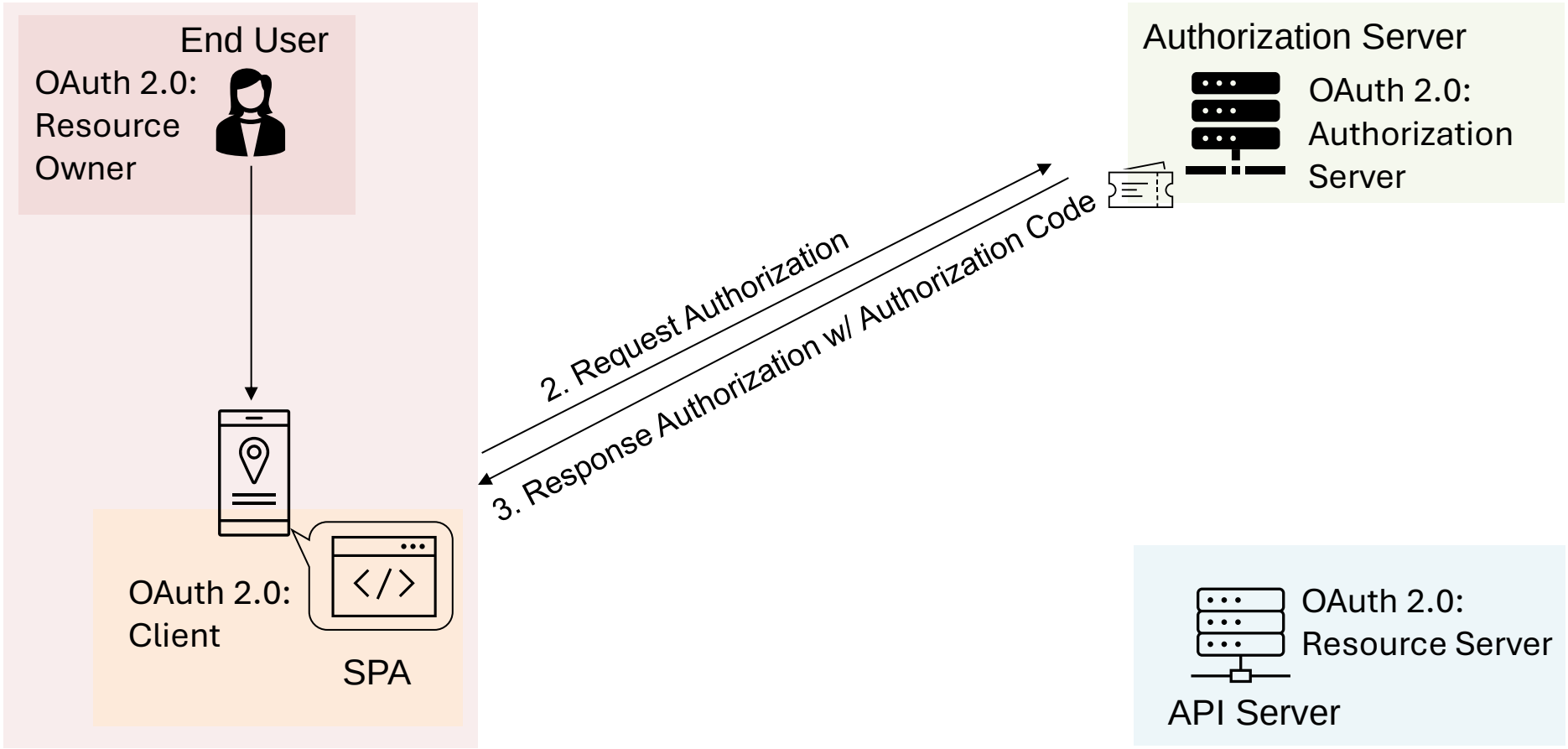
OAuth 2.0:
Authorization
Server



OAuth 2.0:
Resource Server

API Server

SPA in OAuth 2.0

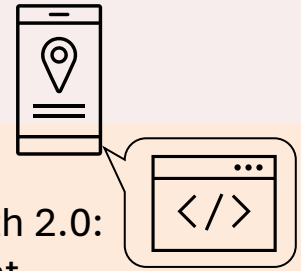


SPA in OAuth 2.0

HITACHI
Inspire the Next

End User

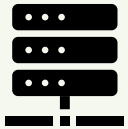
OAuth 2.0:
Resource
Owner



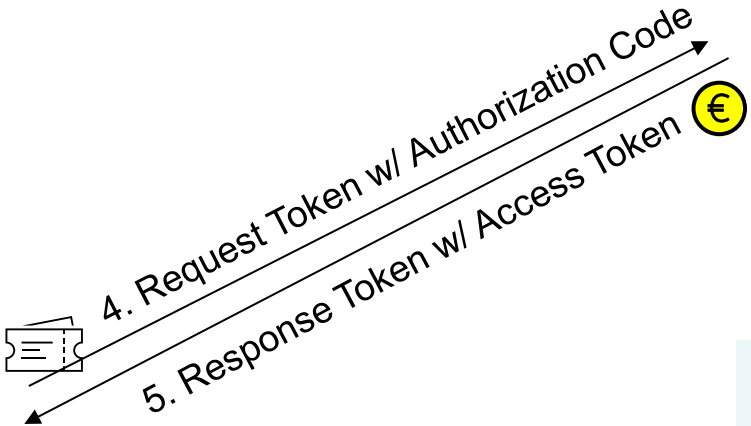
OAuth 2.0:
Client

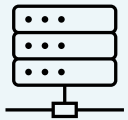
SPA

Authorization Server



OAuth 2.0:
Authorization
Server





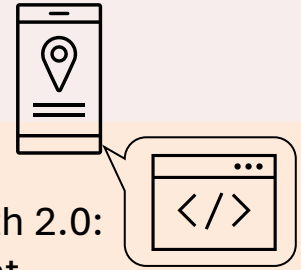
OAuth 2.0:
Resource Server

API Server

SPA in OAuth 2.0

End User

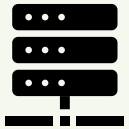
OAuth 2.0:
Resource
Owner



OAuth 2.0:
Client

SPA

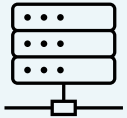
Authorization Server



OAuth 2.0:
Authorization
Server

6. Request API Access w/ Access Token

7. Response API Access

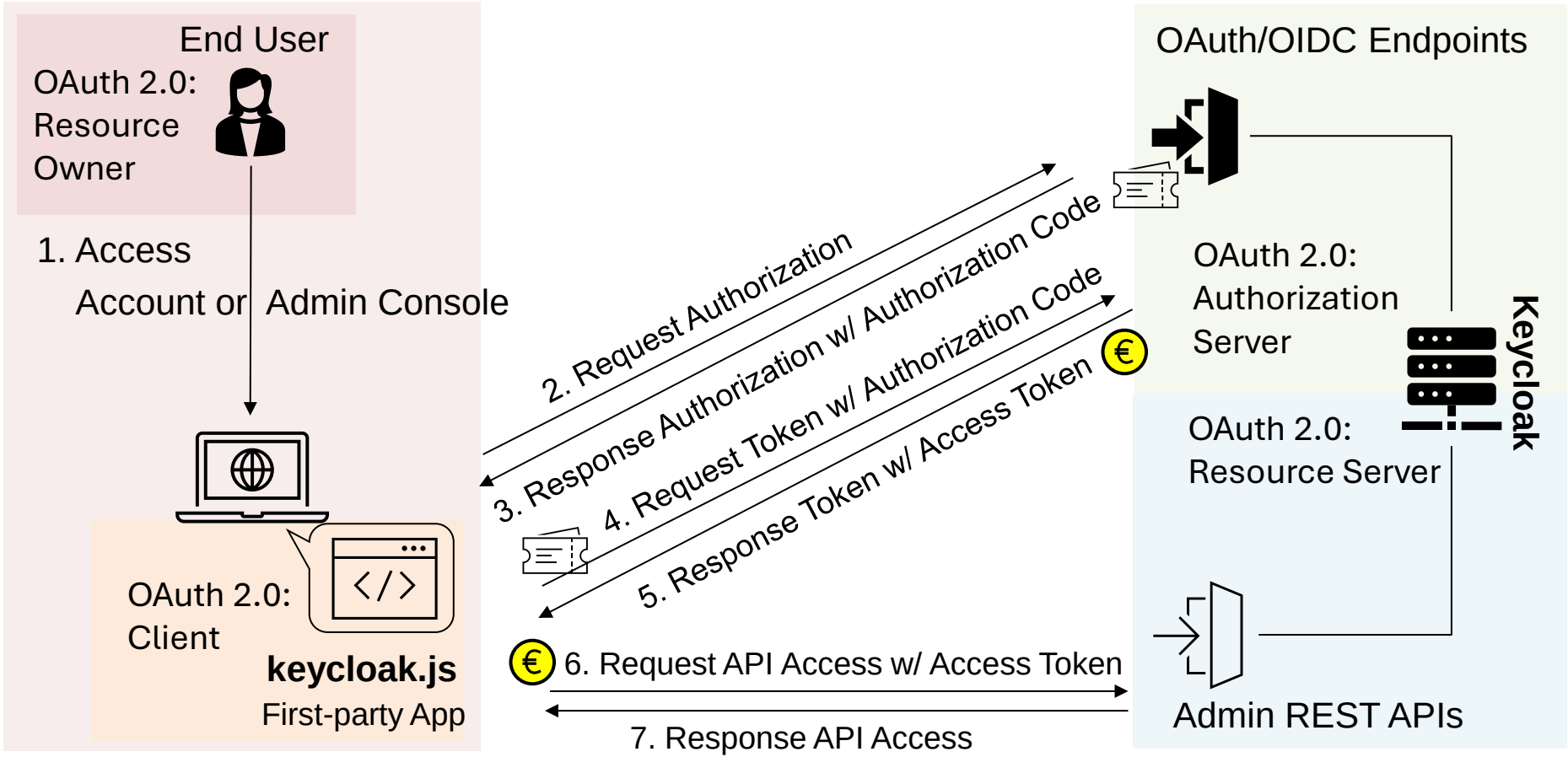


OAuth 2.0:
Resource Server

API Server

SPA in OAuth 2.0 in Practice: Keycloak

HITACHI
Inspire the Next



HITACHI
Inspire the Next

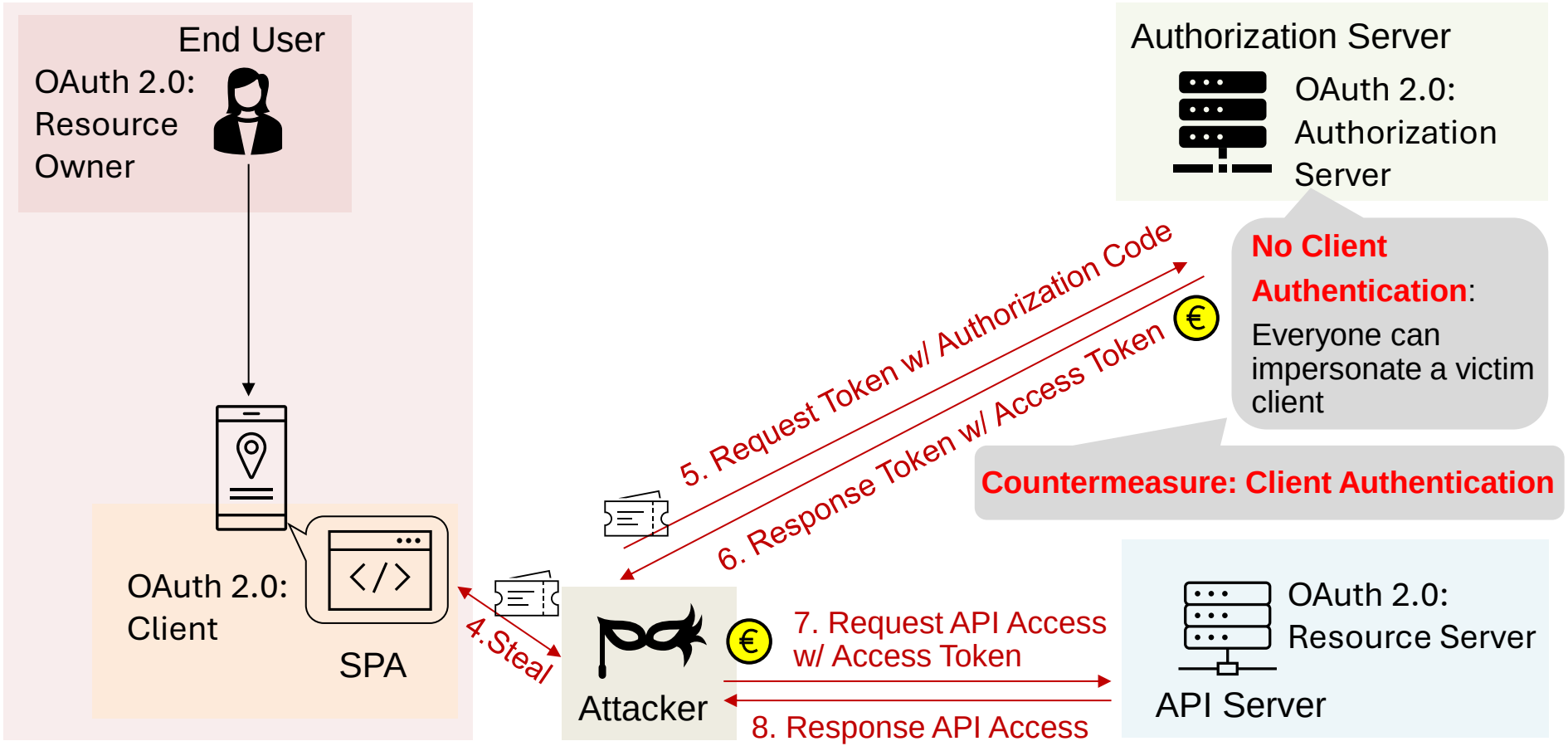
1. SPA in OAuth 2.0

2. Security Risk

3. Countermeasure: DPoP

4. Security Considerations

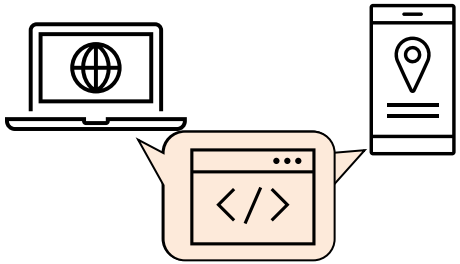
Security Risk: Misuse of a stolen authorization code



Challenge: SPA as OAuth 2.0 Public Client

HITACHI
Inspire the Next

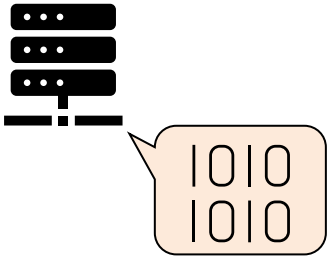
OAuth 2.0 Client Type: Public Client



- Native Application
- User-Agent-Based Application (SPA)

NOT able to do **Client Authentication**:
Incapable of maintaining the confidentiality of credentials and secure client authentication using other means

OAuth 2.0 Client Type: Confidential Client



- Web Application

Able to do **Client Authentication**:
Capable of maintaining the confidentiality of credentials or secure client authentication using other means

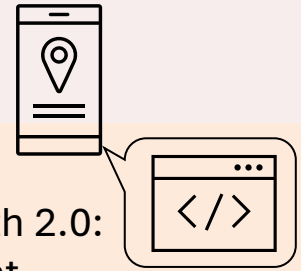
Reference: RFC 6749 OAuth 2.0 (https://datatracker.ietf.org/doc/html/rfc6749)
OAuth 2.0 for Browser-Based Applications
(https://datatracker.ietf.org/doc/html/draft-ietf-oauth-browser-based-apps)

Security Risk: Misuse of a stolen access token

HITACHI
Inspire the Next

End User

OAuth 2.0: Resource Owner



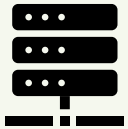
6. Steal

Attacker

7. Request API Access w/ Access Token

8. Response API Access

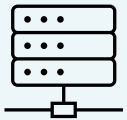
Authorization Server



OAuth 2.0: Authorization Server

Countermeasure: Sender-constraining Token

Bearer Token Usage:
Everyone who has a victim client's access token can use it.



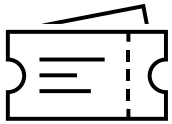
OAuth 2.0: Resource Server

API Server

Sender-constraining Token

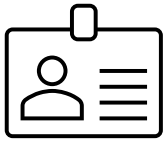
HITACHI
Inspire the Next

Bearer Token(*1)



- Every client who holds can use.
- E.g., train ticket

Sender-constraining Token



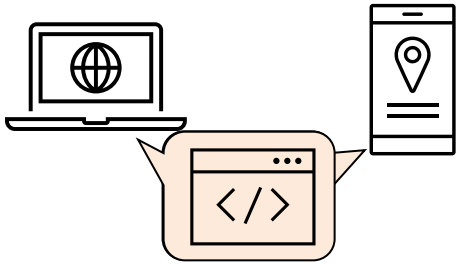
- Only specific client can use.
- E.g., passport
- One of the ways to achieve: OAuth MTLS(*2)
  - Authorization Server binds a token with a X.509 certificate of the client cryptographically.
  - Client needs to securely manage a private key of the certificate with its long lifetime (in years).

Reference:
*1: RFC 6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage
(https://datatracker.ietf.org/doc/html/rfc6750)
*2: RFC 8705 OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens
3. Mutual-TLS Client Certificate-Bound Access Tokens
(https://datatracker.ietf.org/doc/html/rfc8705#name-mutual-tls-client-certifica)

Challenge: SPA as OAuth 2.0 Public Client

HITACHI
Inspire the Next

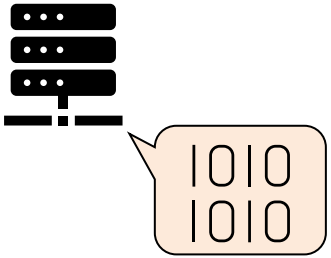
OAuth 2.0 Client Type: Public Client



- Native Application
- User-Agent-Based Application (SPA)

Difficult to use Sender-constraining Token:
Incapable of securely managing a private key of its X.509 certificate with its long lifetime (in years).

OAuth 2.0 Client Type: Confidential Client



- Web Application

Able to use Sender-constraining Token:
Capable of securely managing a private key of its X.509 certificate with its long lifetime (in years).

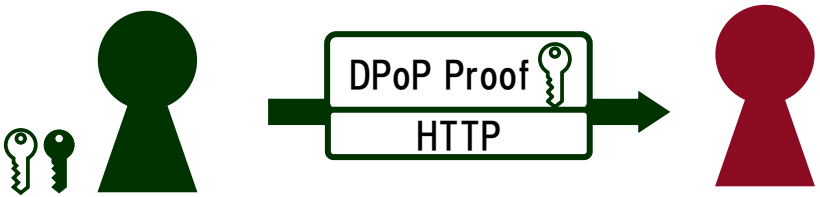
HITACHI
Inspire the Next

1. SPA in OAuth 2.0
2. Security Risk
3. Countermeasure: DPoP
4. Security Considerations

DPoP at a glance

HITACHI
Inspire the Next

OAuth 2.0 Demonstrating Proof of Possession (DPoP)

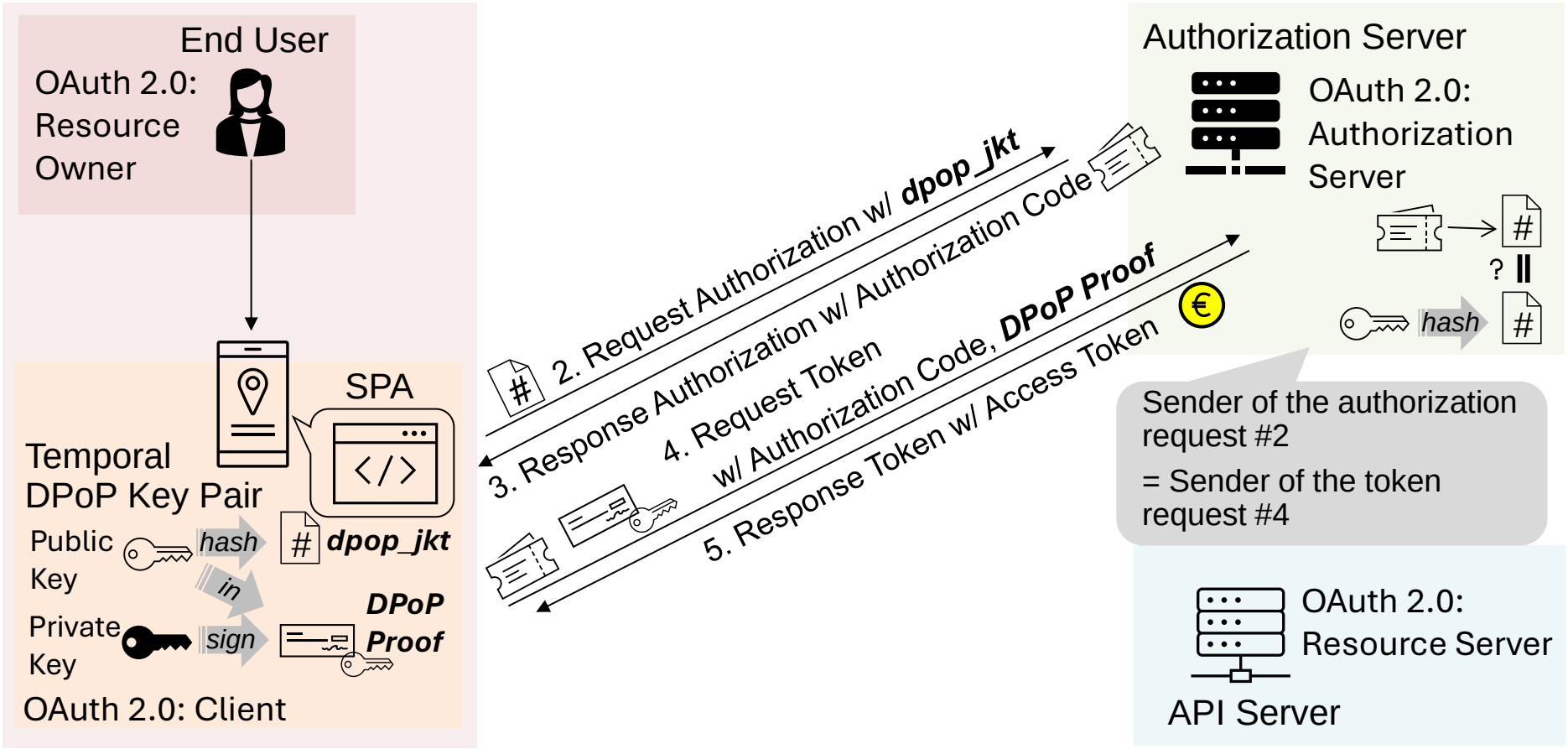


- Demonstrate and verify a sender of a message on HTTP is a holder of an asymmetric cryptography key.
- Defined by RFC 9449[*1].
- It can be used to realize a sender-constraining token by binding a key with a token in a cryptographic way.

*1: https://datatracker.ietf.org/doc/html/rfc9449

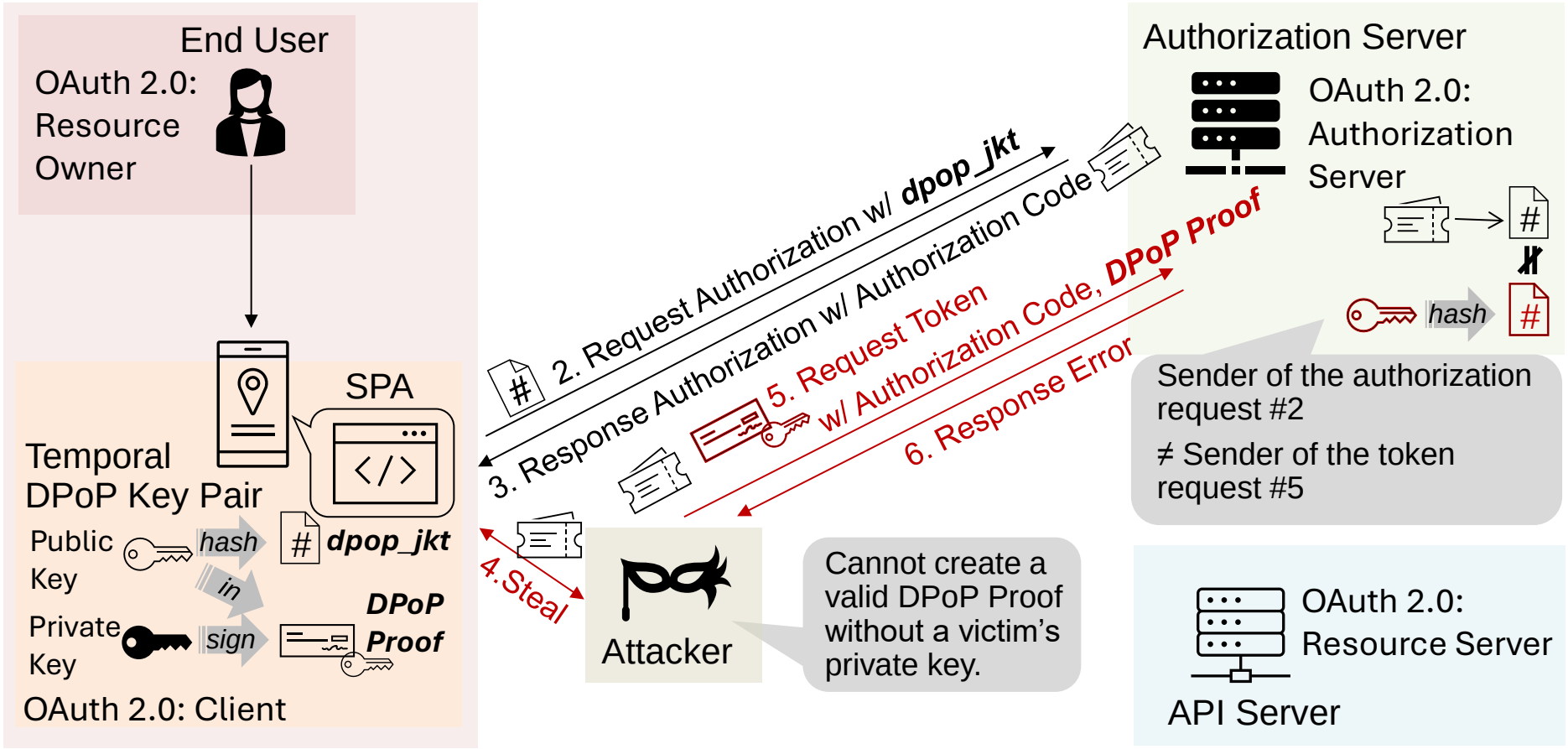
Misuse of a stolen authorization code : Authorization Code Binding to a DPoP Key

HITACHI
Inspire the Next



Misuse of a stolen authorization code : Authorization Code Binding to a DPoP Key

HITACHI
Inspire the Next



Misuse of a stolen access token: Sender-constraining Access Token by DPoP

HITACHI
Inspire the Next

End User

OAuth 2.0: Resource Owner

SPA

Temporal DPoP Key Pair

Public Key

Private Key

in

sign

DPoP Proof

OAuth 2.0: Client

Authorization Server

OAuth 2.0: Authorization Server

Bind

4. Request Token w/ Authorization Code, ***DPoP Proof***

5. Response Token w/ Access Token

Sender of the token request #4

= Sender of the API request #6

6. Request API Access w/ Access Token, ***DPoP Proof***

7. Response API Access

OAuth 2.0: Resource Server

API Server

Misuse of a stolen access token: Sender-constraining Access Token by DPoP

HITACHI
Inspire the Next

End User
OAuth 2.0: Resource Owner

SPA
Temporal DPoP Key Pair
Public Key
hash
in
Private Key
sign
dpop_jkt
DPoP Proof
OAuth 2.0: Client

Authorization Server
OAuth 2.0: Authorization Server
Bind

4. Request Token w/ Authorization Code, DPoP Proof
5. Response Token w/ Access Token

Sender of the token request #4
≠ Sender of the API request #7

Cannot create a valid DPoP Proof without a victim's private key.

6. Steal
Attacker
7. Request API Access w/ Access Token, DPoP Proof
8. Response Error

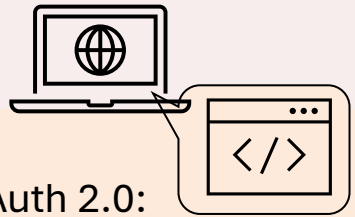
OAuth 2.0: Resource Server
API Server

Demo 1

HITACHI
Inspire the Next

End User

OAuth 2.0:
Resource
Owner



OAuth 2.0:
Client

Vue +
Panva openid-client

1.

2.

OAuth 2.0:
Authorization Server

Keycloak

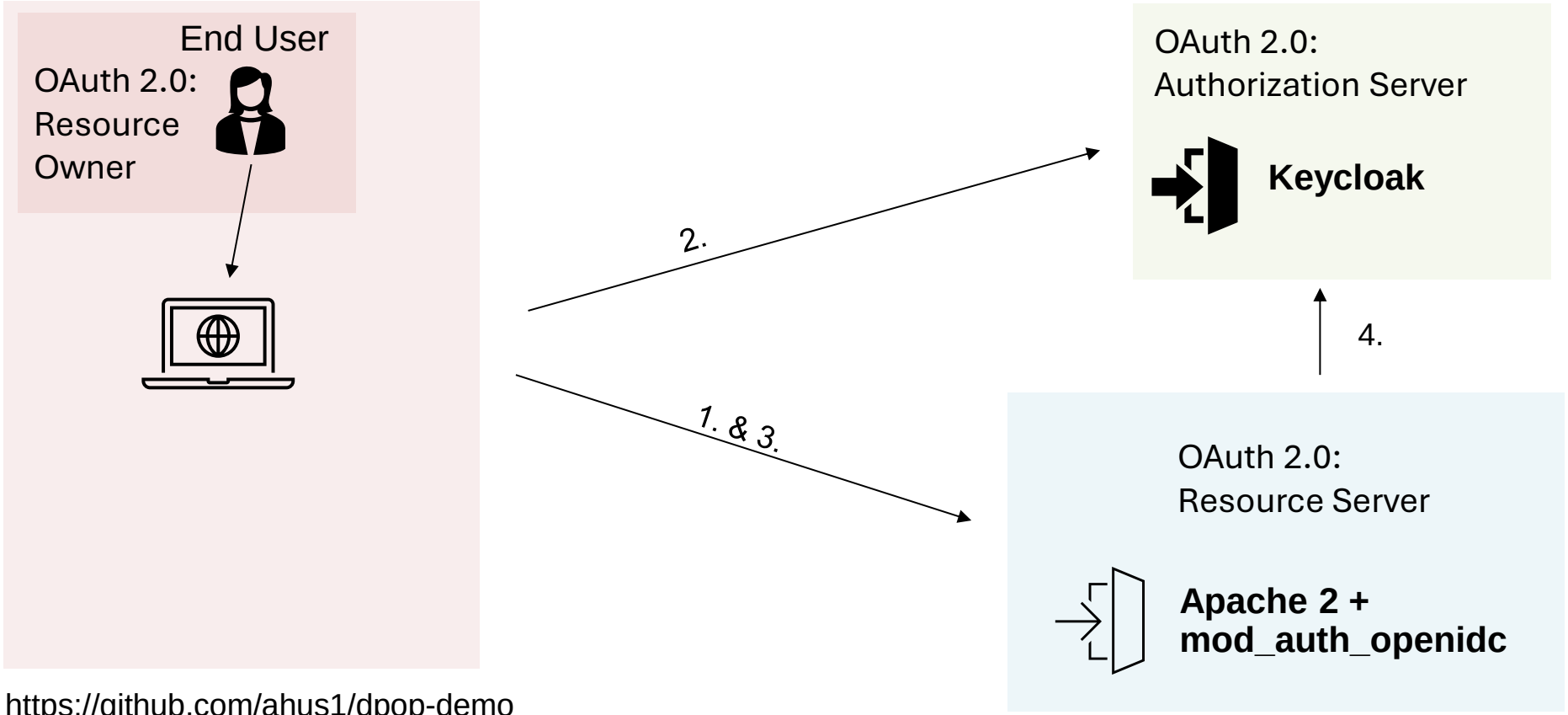
3.

OAuth 2.0:
Resource Server

Quarkus +
Nimbus OAuth 2.0 SDK

https://github.com/ahus1/dpop-demo

Demo 2



https://github.com/ahus1/dpop-demo

HITACHI
Inspire the Next

1. SPA in OAuth 2.0
2. Security Risk
3. Countermeasure: DPoP
4. Security Considerations

Security Considerations: Defense in Depth

HITACHI
Inspire the Next

- Both countermeasures are needed
  1. Misuse of a stolen authorization code
  2. Misuse of a stolen access token

  If 1 lacks, an attacker stealing an authorization code can receive an access token by using the attacker's DPoP key pairs and DPoP proof. The access token is bound with attacker's DPoP key.

- Several countermeasures of misuse of a stolen authorization code
  - Proof Key for Code Exchange (PKCE)*1
  - DPoP dpop_jkt

  PKCE is also an effective counter measure for the misuse.
  IMO, supporting PKCE is easier than DPoP dpop_jkt.

*1: RFC 7636 Proof Key for Code Exchange by OAuth Public Clients

Security Considerations: Defense in Depth

HITACHI
Inspire the Next

Options of countermeasure combination

Misuse of a stolen authorization code		Misuse of a stolen access token	Notes
PKCE	DPoP dpop_jkt	DPoP	
-	-	-	
-	-	x	
x	-	x	Recommended
-	x	x	Recommended
x	x	x	Recommended

Summary

- SPA in OAuth 2.0, there are some risks: misuse of a stolen authorization code, misuse of a stolen access token.
- DPoP is promising countermeasure for the risks.
- To use DPoP securely, need to take case of some points.

You can download the slides here 👆

Trademarks

HITACHI
Inspire the Next

- GitHub is a trademark or registered trademark of GitHub, Inc. in the United States and other countries.
- Other brand names and product names used in this material are trademarks, registered trademarks, or trade names of their respective holders.

Appendix

Security Considerations: Lifecycle Management of DPoP Key Pairs

HITACHI
Inspire the Next

Relatively short lifetime of DPoP key pairs, so SPA can retain them on its memory.

- When generating DPoP key pairs
  - Before crafting an authorization request (if using dpop_jkt) OR
  - Before crafting a token request

- When deleting DPoP key pairs
  - When an access token expired / revoked OR
  - When a refresh token expired / revoked (if a refresh token bound with DPoP Key*1)

- Where DPoP key pairs are stored
  - Secure storage (e.g., WebCrypto*2)

- Re-use of DPoP key pairs
  - Not recommended.

*1: RFC 9449 DPoP describes it for a public client (due to incapable of client authentication)

*2: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API

Security Considerations: Replay Attack

HITACHI
Inspire the Next

- Countermeasure
  - Scoping DPoP proof narrowly : “htm” and “htu”
    “htm”: HTTP method for sending DPoP Proof, “htu”: URI where DPoP Proof is sent
    ⇒ An attacker cannot replay the DPoP proof to other URI in other HTTP method.
  - Unique ID : “jti”
    ⇒ A receiver of DPoP proofs can detect multiple use of the same DPoP proof
  - Short lifetime : “iat”
    ⇒ A receiver can reject a DPoP proof if the current time is away from “iat” value.

Example: DPoP Proof’s Body (Token Request)

```
{
  "htm": "POST",
  "htu": "https://as.keycloak-
fapi.org/auth/realms/test/protocol/openid-
connect/token",
  "iat": 1733100119,
  "jti": "LUzr1mmDT4bN9ug1D5rf"
}
```

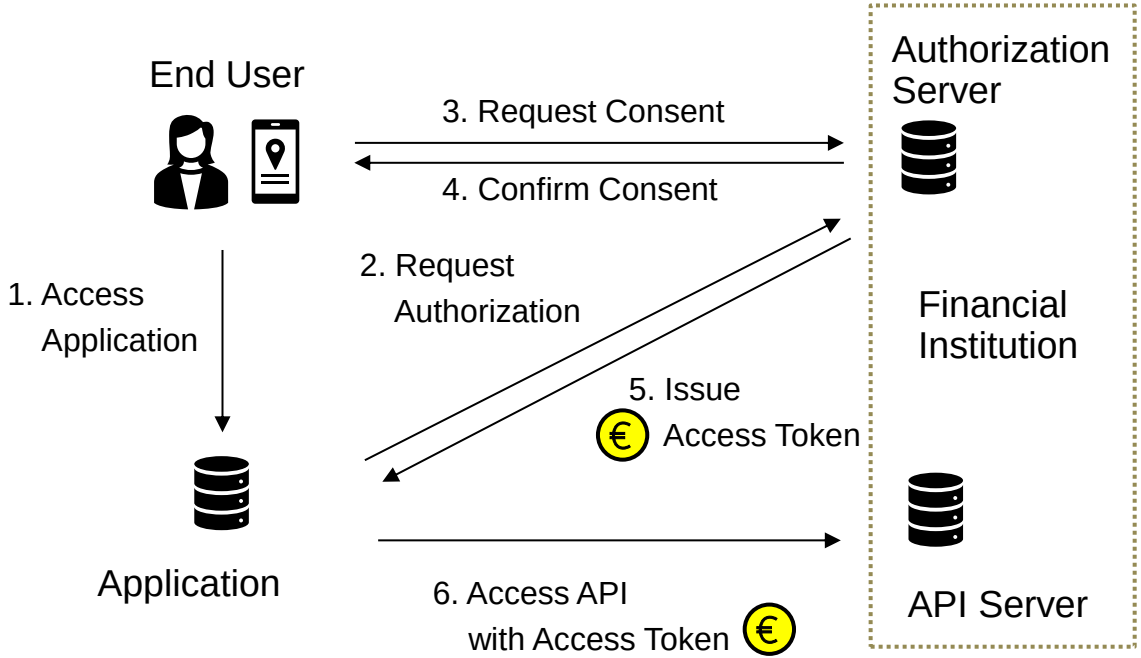
DPoP Use-case

HITACHI
Inspire the Next

■ FAPI 2.0 Security Profile for Open Banking

Open Banking:

- A financial institution provides its financial services via APIs.
- A third-party application accesses the APIs on behalf of an end user.



FAPI 2.0 Security Profile:

- A security specification for securing API access.
- It is intended to be used for a use-case that requires high level of security for API access like Open Banking.
- FAPI 2.0 adopts **DPoP** as one option for sender-constraining token.

DPoP Libraries

HITACHI
Inspire the Next

- openid-client (https://github.com/panva/openid-client)
  - OAuth 2 / OpenID Connect Client API for JavaScript Runtimes
  - It is certified by OpenID Foundation that it complies with FAPI 2.0 (https://openid.net/certification/)

- mod_auth_openidc (https://github.com/OpenIDC/mod_auth_openidc)
  - OAuth 2 / OpenID Connect for Apache 2.x HTTPD server
  - It is certified by OpenID Foundation that is complies with Relying Party (https://openid.net/certification/)

- Nimbus OAuth 2.0 SDK (https://connect2id.com/products/nimbus-oauth-openid-connect-sdk)
  - OAuth 2 / OpenID Connect for Java
  - Framework-agnostic

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: Authorization Request

Parameter	Value
client_id	client1-dpop-mtls-ES256-ES256-fapi2-security-profile
redirect_uri	https://conformance-suite.keycloak-fapi.org/test/a/keycloak/callback
scope	email profile
state	GXOwomY5tq
response_type	code
code_challenge	Vgl3b6JnsZ3LE6bHBRut2GPqYFYKDnFXQJoxVOCIQ3A
code_challenge_method	S256
dpop_jkt	OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: Token Request

Header	Value
Accept	application/json
Content-Type	application/x-www-form-urlencoded;charset=UTF-8
Content-Length	451
DPoP	eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6IkVDIiwidXNlIjoic2lnIiwiY3J2IjoiUC0yNTYiLCJraWQiOiJPWC0tS3hCbGYzNGU0S2RQazRmU3ZPSzFzbkZhZ3laZERTTjhiSHEwdGk0IiwieCI6IkMxMlFYR3l3T3gweGVaM3VLTVlyV2NuTW80WHVRVHRnS0pDT3ZCdFdmTUEiLCJ5IjoiY3hkcmJyd2l3V0dPUG92SXBfNUFLUlo4eEtrdU5GVjBKNWEwU2FDejlBOCIsImFsZyI6IkVTMjU2In19.eyJodG0iOiJQT1NUIiwiaHR1IjoiaHR0cHM6Ly9hcy5rZXljbG9hay1mYXBpLm9yZy9hdXRoL3JlYWxtcy90ZXN0L3Byb3RvY29sL29wZW5pZC1jb25uZWN0L3Rva2VuIiwiaWF0IjoxNzMzMTExNzY5LCJqdGkiOiJzQ3pqNXBjUTJpakJVdGJDODNxVyJ9.cI_PnOwjasVwdRKi5m5wcpfl-LhxAndRfUIFF9pLHcR1YVIWUOu8ZO3lIPtnV0kLG1Ovek3-4GsbQx8-zk6dYQ

DPoP Proof (base64-url encoded signed JSON Web Token (JWT)) :=

< JSON Object Signing and Encryption (JOSE) Header>.<Body>.<signature>

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: DPoP Proof's JOSE Header (Token Request)

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "kid": "OX--
KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4",
    "x": "C12QXGywOx0xeZ3uKMYrWcnMo4XuQTtgKJCOvBtWfMA",
    "y": "cxdrbrwiwWGOPovIp_5AKRZ8xKkuNFV0J5a0SaCz9A8",
    "alg": "ES256"
  }
}
```

Example: DPoP Proof's Body (Token Request)

```
{
  "htm": "POST",
  "htu": "https://as.keycloak-
fapi.org/auth/realms/test/protocol/openid-
connect/token",
  "iat": 1733111769,
  "jti": "sCzj5pcQ2ijBUtbC83qW"
}
```

EC Key's public key

SHA-256 Hash

`OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4`

Same

Example: dpop_jkt (Authorization Request)

dpop_jkt	OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4

Countermeasure: DPoP

Example: Token Request

request_uri	https://as.keycloak-fapi.org/auth/realms/test/protocol/openid-connect/token
request_method	POST

Same

Example: DPoP Proof's Body (Token Request)


```
{
  "htm": "POST",
  "htu": "https://as.keycloak-fapi.org/auth/realms/test/protocol/openid-connect/token",
  "iat": 1733111769,
  "jti": "sCzj5pcQ2ijBUtbC83qW"
}
```

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: DPoP Proof's JOSE Header (Token Request)

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "kid": "OX--
KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4",
    "x": "C12QXGywOx0xeZ3uKMYrWcnMo4XuQTtgKJCOvBtWfMA",
    "y": "cxdrbrwiwWGOPovIp_5AKRZ8xKkuNFV0J5a0SaCz9A8",
    "alg": "ES256"
  }
}
```

EC public key

Example: Access Token (Token Response)

```
  ],
  "cnf": {
    "jkt": "OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4"
  },
  "scope": "profile email",
  "email_verified": false,
  "preferred_username": "john"
}
```

Same

SHA-256 Hash

OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4

Same

Example: dpop_jkt (Authorization Request)

dpop_jkt	OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: API Request

Header	Value
Accept	application/json
Content-Length	0
Authorization	DPoP eyJhbGciOiJQUz … partially omitted … rJUQgMaMg
DPoP	eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6IkVDIiwidXNlIjoic2lnIiwiY3J2IjoiUC0yNTYiLCJraWQiOiJPWC0tS3hCbGYzNGU0S2RQazRmU3ZPSzFzbkZhZ3laZERTTjhiSHEwdGk0IiwieCI6IkMxMlFYR3l3T3gweGVaM3VLTVlyV2NuTW80WHVRVHRnS0pDT3ZCdFdmTUEiLCJ5IjoiY3hkcmJyd2l3V0dPUG92SXBfNUFLUlo4eEtrdU5GVjBKNWEwU2FDejlBOCIsImFsZyI6IkVTMjU2In19.eyJodG0iOiJHRVQiLCJodHUiOiJodHRwczovL3JzLmtleWNsb2FrLWZhcGkub3JnL2Rwb3AiLCJhdGgiOiIwenFfb0JvMS1qeHJlTDZDODFpWUZ6dENUMlA2LTlkMXBqN0NVX3g0VjhNIiwiaWF0IjoxNzMzMTExNzY5LCJqdGkiOiJuMDNwT09SRFU4TEhwWW5BbVUzSyJ9.2f7xfeT-VsXCY4qGwN5mYHfXua1a9ybKXSMZ56DvzwINhwU8WriRYOxvDn2bxRugq8OqahYGBge5Qjxx5nrNiA

DPoP Proof →

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: DPoP Proof's JOSE Header (API Request)

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "kid": "OX--
KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4",
    "x": "C12QXGywOx0xeZ3uKMYrWcnMo4XuQTtgKJCOvBtWfMA",
    "y": "cxdrbrwiwWGOPovIp_5AKRZ8xKkuNFV0J5a0SaCz9A8",
    "alg": "ES256"
  }
}
```

Example: Access Token (API Request)

```
  ],
  "cnf": {
    "jkt": "OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4"
  },
  "scope": "profile email",
  "email_verified": false,
  "preferred_username": "john"
}
```

EC public key

SHA-256 Hash

Same

`OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4`

Same

Example: dpop_jkt (Authorization Request)

dpop_jkt	OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: DPoP Proof's JOSE Header (API Request)

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "kid": "OX--
KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4",
    "x": "C12QXGywOx0xeZ3uKMYrWcnMo4XuQTtgKJCOvBtWfMA",
    "y": "cxdrbrwiwWGOPovIp_5AKRZ8xKkuNFV0J5a0SaCz9A8",
    "alg": "ES256"
  }
}
```

Example: DPoP Proof's JOSE Header (Token Request)

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "kid": "OX--
KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4",
    "x": "C12QXGywOx0xeZ3uKMYrWcnMo4XuQTtgKJCOvBtWfMA",
    "y": "cxdrbrwiwWGOPovIp_5AKRZ8xKkuNFV0J5a0SaCz9A8",
    "alg": "ES256"
  }
}
```

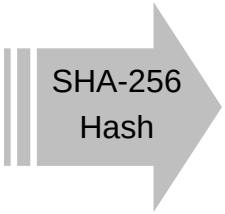
Same EC Key's public key

Countermeasure: DPoP

HITACHI
Inspire the Next

Example: Access Token (API Request)

```
  ],
  "cnf": {
    "jkt": "OX--KxBlf34e4KdPk4fSvOK1snFagyZdDSN8bHq0ti4"
  },
  "scope": "profile email",
  "email_verified": false,
  "preferred_username": "john"
}
```




```
0zq_oBo1-jxreL6C81iYFztCT2P6-
9d1pj7CU_x4V8M
```

Example: DPoP Proof's JOSE Header (API Request)

```
{
  "htm": "GET",
  "htu": "https://rs.keycloak-fapi.org/dpop",
  "ath": "0zq_oBo1-jxreL6C81iYFztCT2P6-9d1pj7CU_x4V8M",
  "iat": 1733111769,
  "jti": "n03pOORDU8LHpYnAmU3K"
}
```

Same

Using DPoP to use access tokens securely in your Single Page Applications

Université Libre de Bruxelles Campus du Solbosch, Brussels, Belgium

1 February 2025 

Alexander Schwartz @ Red Hat
Takashi Norimatsu @ Hitachi, Ltd.

END