EXTENSION RFCS

PUBLISHED RFCS – BASIC

- OAuth 2.0 Threat Model and Security Considerations (RFC6819)
- JSON Web Token (JWT) (RFC 7519)

PUBLISHED RFCS – AUTHENTICATION&AUTHORIZATION

- Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants (RFC 7521)

- Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants (RFC 7522)
- JSON Web Token (JWT) Profile For OAuth 2.0 Client Authentication and Authorization Grants (RFC 7523)

RFC7521 -ASSERTION FRAMEWORK FOR OAUTH 2.0 CLIENT AUTHENTICATION AND AUTHORIZATION GRANTS

- Using Assertions as Authorization Grants
  - grant_type
  - assertion
  - scope

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&
assertion=PHNhbWxwOl...[omitted for brevity]…ZT4
```

- Using Assertions for Client Authentication
  - client_assertion_type
  - client_assertion
  - client_id

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=n0esc3NRze7LTCu7iYzS6a5acc3f0ogp4&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer&
client_assertion=PHNhbW...[omitted for brevity]...ZT
```

RFC 7523– JSON WEB TOKEN (JWT) PROFILE FOR OAUTH 2.0 CLIENT AUTHENTICATION AND AUTHORIZATION GRANTS

- This specification defines the use of a JSON Web Token (JWT) Bearer Token as a means for requesting an OAuth 2.0 access token as well as for client authentication.

PUBLISHED RFCS – DYNAMIC CLIENT REGISTRATION

- OAuth 2.0 Dynamic Client Registration Protocol (RFC 7591)
- OAuth 2.0 Dynamic Client Registration Management Protocol (RFC 7592 EXP)

PUBLISHED RFCS – TOKEN HANDLING

- OAuth2 Token Revocation (RFC 7009)
- OAuth 2.0 Token Introspection (RFC 7662)

PUBLISHED RFCS – SECURITY

- Proof Key for Code Exchange by OAuth Public Clients (RFC 7636)
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) (RFC 7800)

PKCE (RFC 7636)

A. The client creates and records a secret named the ”code_verifier" and derives a transformed version "t(code_verifier)" (referred to as the "code_challenge"), which is sent in the OAuth 2.0 Authorization Request along with the transformation method ”t_m".

B. The Authorization Endpoint responds as usual but records "t(code_verifier)" and the transformation method.

C. The client then sends the authorization code in the Access Token Request as usual but includes the "code_verifier" secret generated at (A).

D. The authorization server transforms "code_verifier" and compares it to "t(code_verifier)" from (B). Access is denied if they are not equal.

PKCE - EXAMPLE

Setup:

code_verifier=dBjftJeZ4CVPmB92K27uhbUJU1p1r_wW1gFWFOEjXk
code_challenge_method=S256

code_challenge_method(code_verifier) = code_challenge

code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw

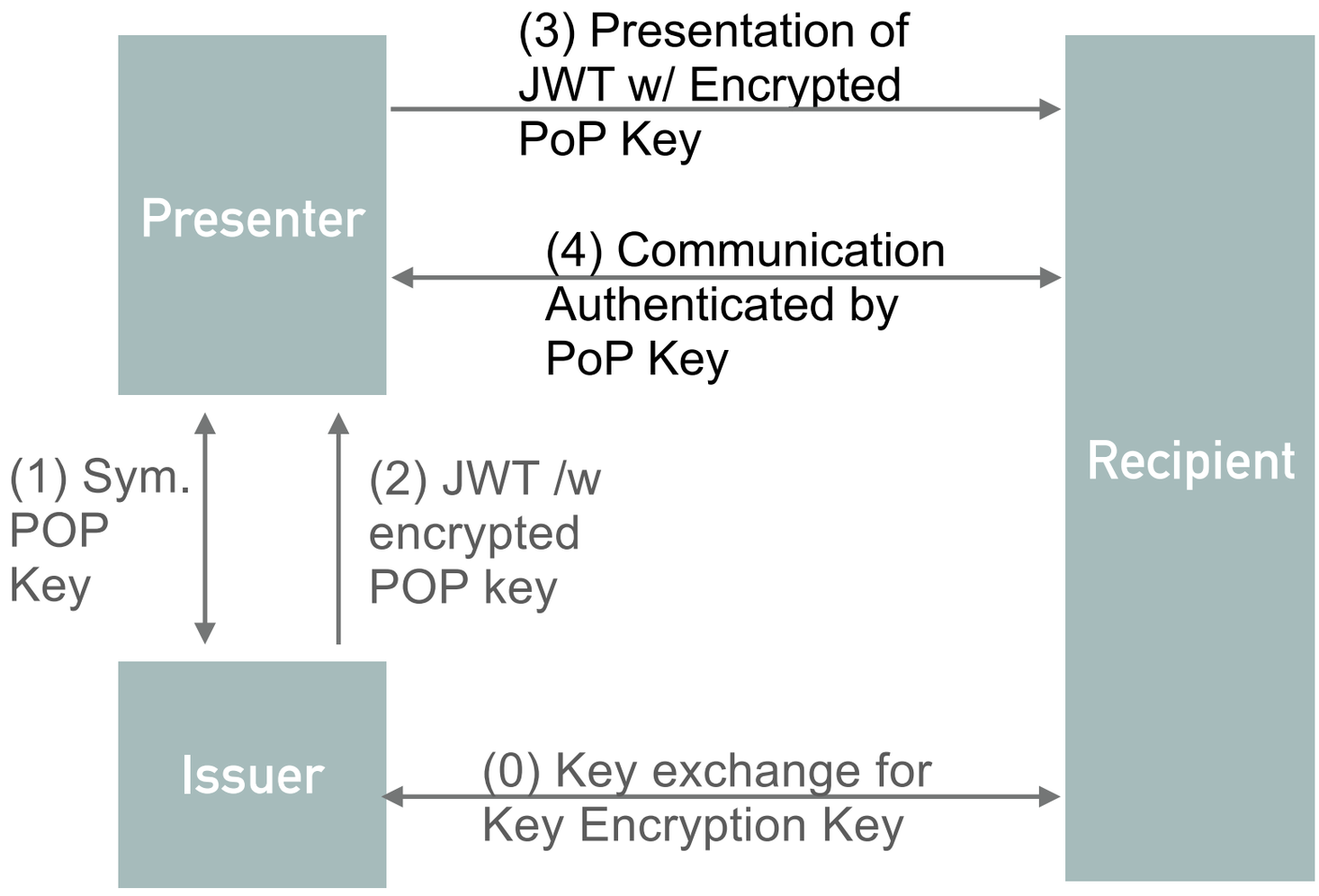
Authorization request:

```
code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM&code_challenge_method=S256
```

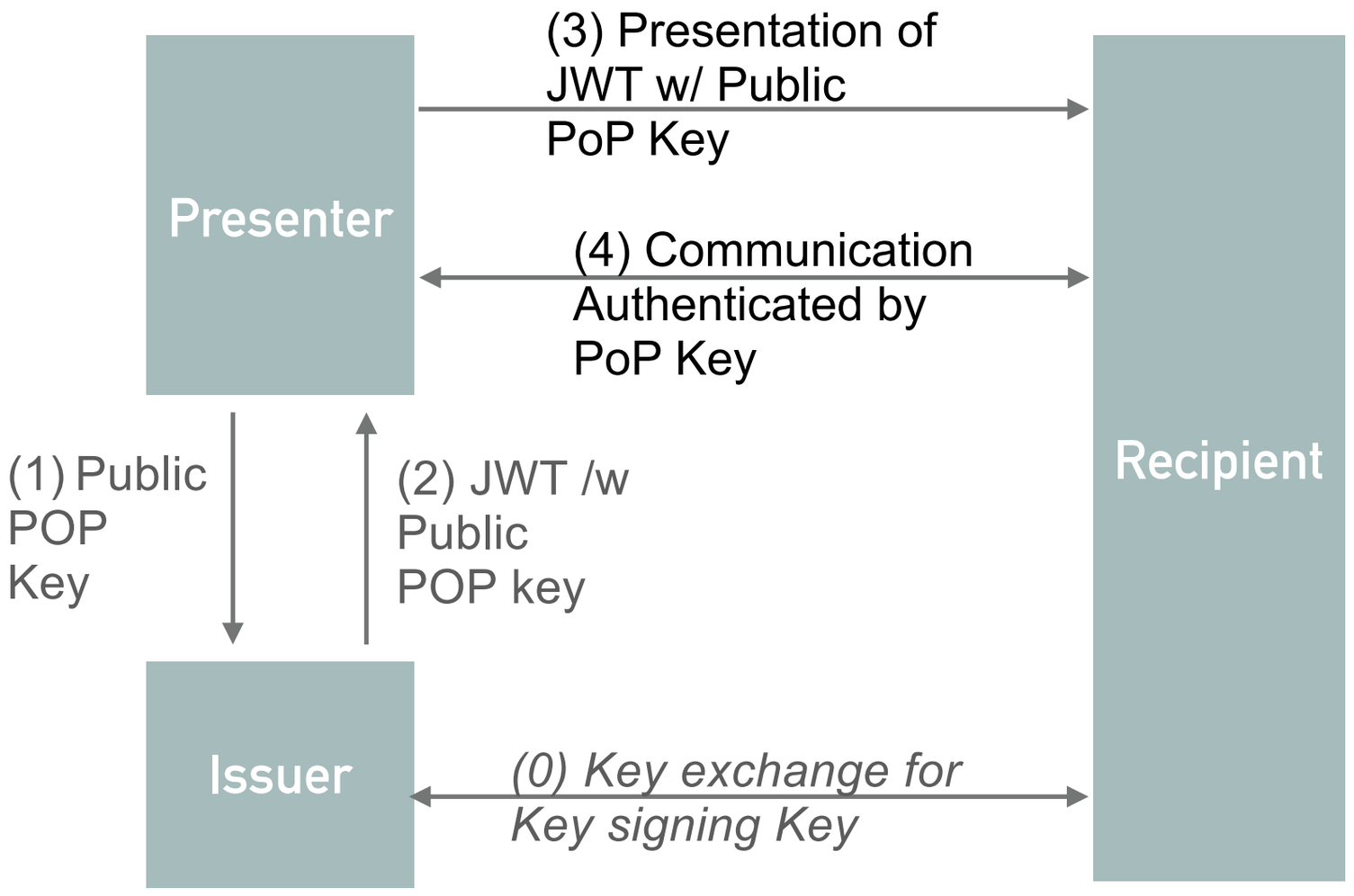
Token request:

```
code_verifier=dBjftJeZ4CVPmB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

RFC7800 PROOF OF POSSESSION WITH SYMMETRIC KEYS



RFC7800 POP WITH ASYMMETRIC KEY



RFC7800 CONFIRMATION CLAIM

Encrypted Symmetric

```
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "nonce": "n-0S6_WzA2Mj",
   "exp": 1311281970,
   "iat": 1311280970,
   "cnf":{
      "jwe":
         "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhDQkMtSFMyNTYifQ.
          (remainder of JWE omitted for brevity)"
   }
}
```

Aymmetric

```
{
   "iss": "https://server.example.com",
   "aud": "https://client.example.org",
   "exp": 1361398824,
   "cnf":{
      "jwk":{
         "kty": "EC",
         "use": "sig",
         "crv": "P-256",
         "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
         "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
      }
   }
}
```

KID

```
{
   "iss": "https://server.example.com",
   "aud": "https://client.example.org",
   "exp": 1361398824,
   "cnf":{
      "kid": "dfd1aa97-6d8d-4575-a0fe-34b96de2bfad"
   }
}
```

URL

```
{
   "iss": "https://server.example.com",
   "sub": "17760704",
   "aud": "https://client.example.org",
   "exp": 1440804813,
   "cnf":{
      "jku": "https://keys.example.net/pop-keys.json",
      "kid": "2015-08-28"
   }
}
```

WORKING GROUP DRAFTS

WORKING GROUP DRAFTS – RFC-EDITOR'S QUEUE

- draft-ietf-oauth-amr-values
  Authentication Methods References

WORKING GROUP DRAFTS – IESG PROCESSING

- draft-ietf-oauth-discovery
- draft-ietf-oauth-jwsreq

  Request parameters in a JSON Web Token (JWT)
- draft-ietf-oauth-native-apps

  Authorization requests from native apps should only be made through external user-agents, primarily the user's browser.

WORKING GROUP DRAFTS – ACTIVE

- draft-ietf-oauth-device-flow

  The device flow is suitable for OAuth 2.0 clients executing on devices that do not have an easy data-entry method

- draft-ietf-oauth-mlts

  Mutual TLS profiles for OAuth clients

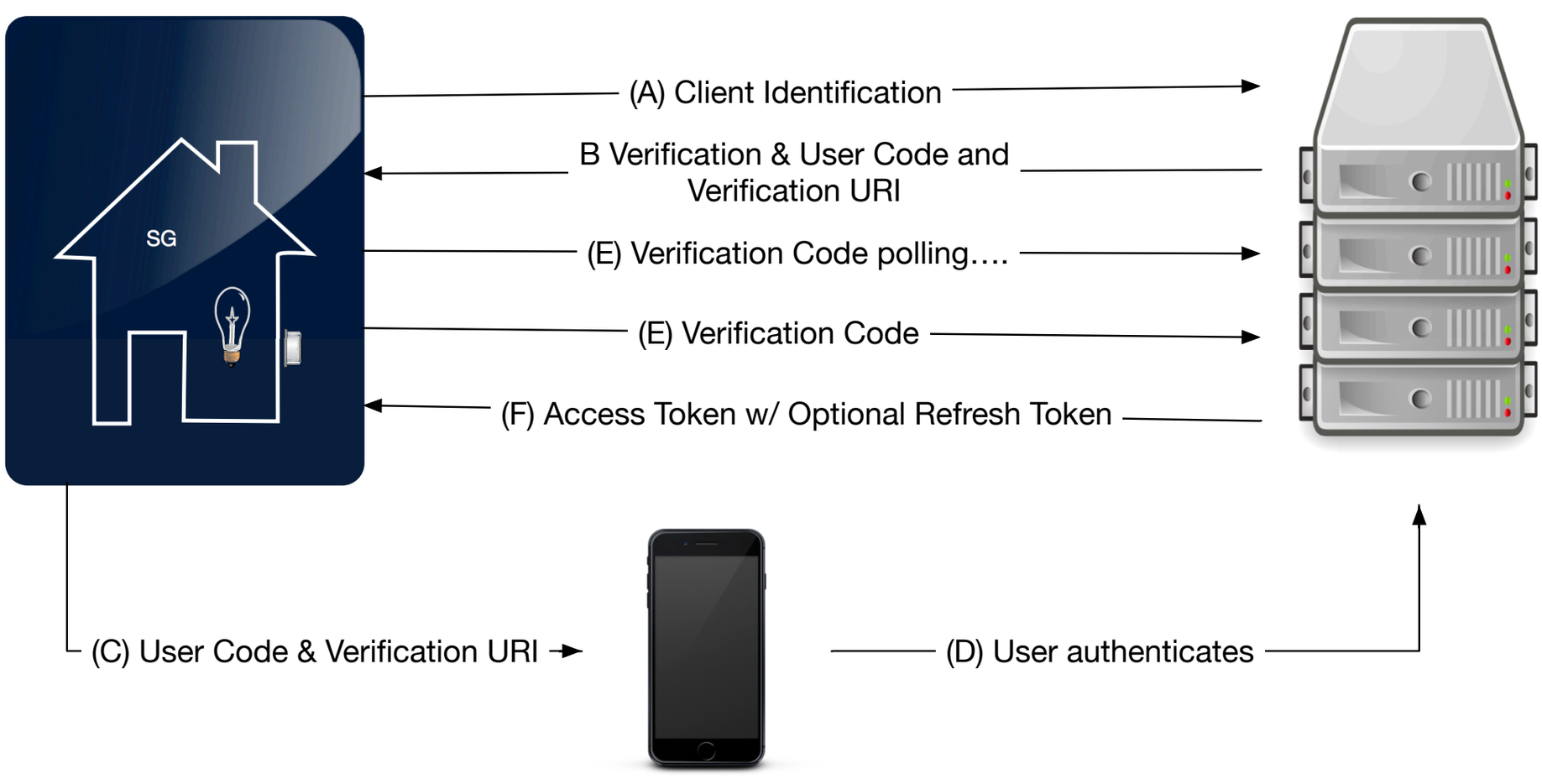
- draft-ietf-oauth-pop-key-distribution

  Proof-of-Possession: AS to client key distribution

- draft-ietf-oauth-security-topics
- draft-ietf-oauth-token-binding
- draft-ietf-oauth-token-exchange

DRAFT-IETF-OAUTH-DEVICE-FLOW

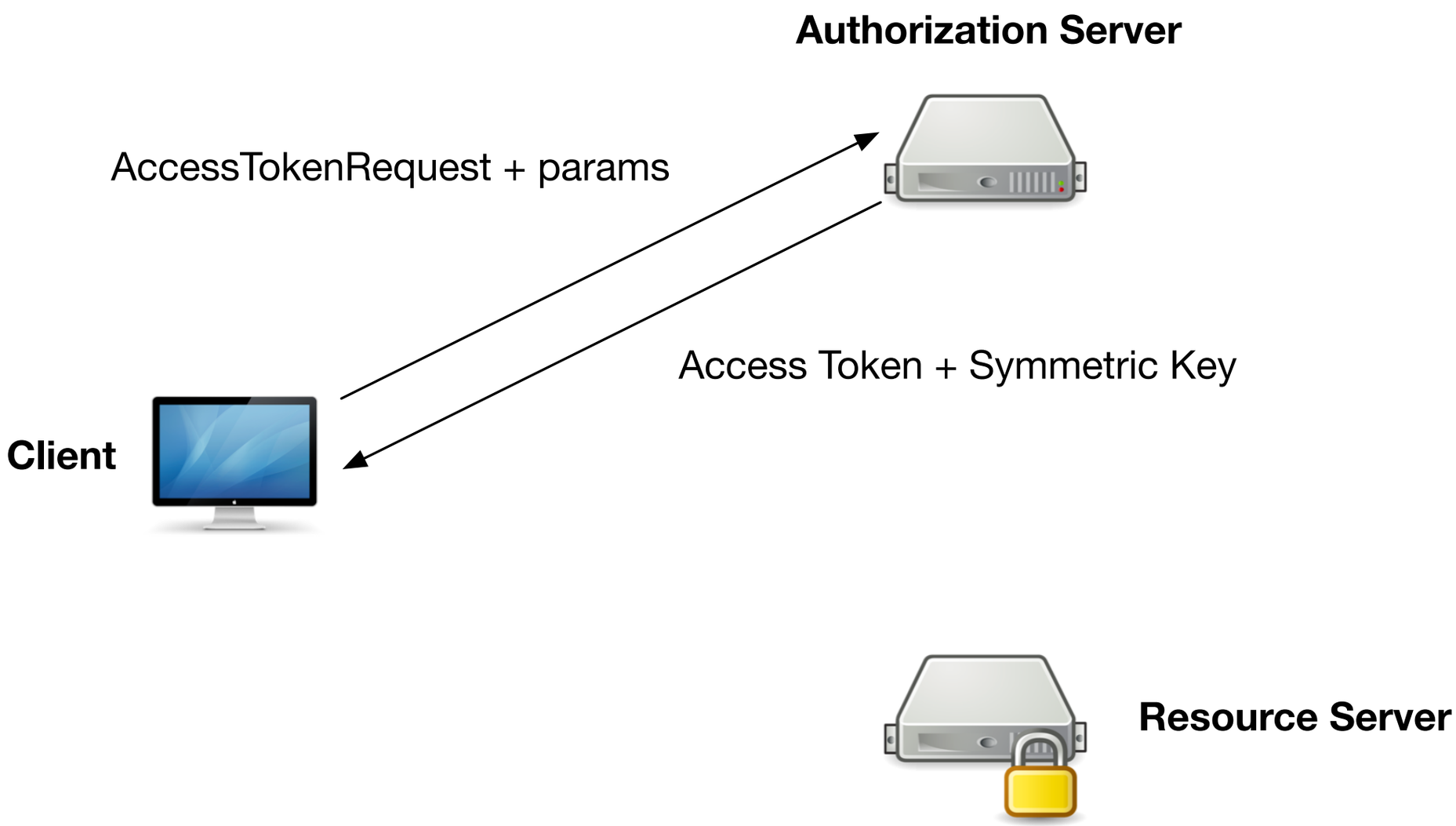
- OAuth 2.0 clients executing on devices that do not have an easy data-entry method
- end-user has separate access to a user-agent on another computer or device



POP SECURITY ARCHITECTURE

- Some scenarios demand additional security protection whereby a client needs to demonstrate possession of cryptographic keying material when accessing a protected resource.

[POP] CLIENT <-> AS INTERACTION (SYMMETRIC KEYS)



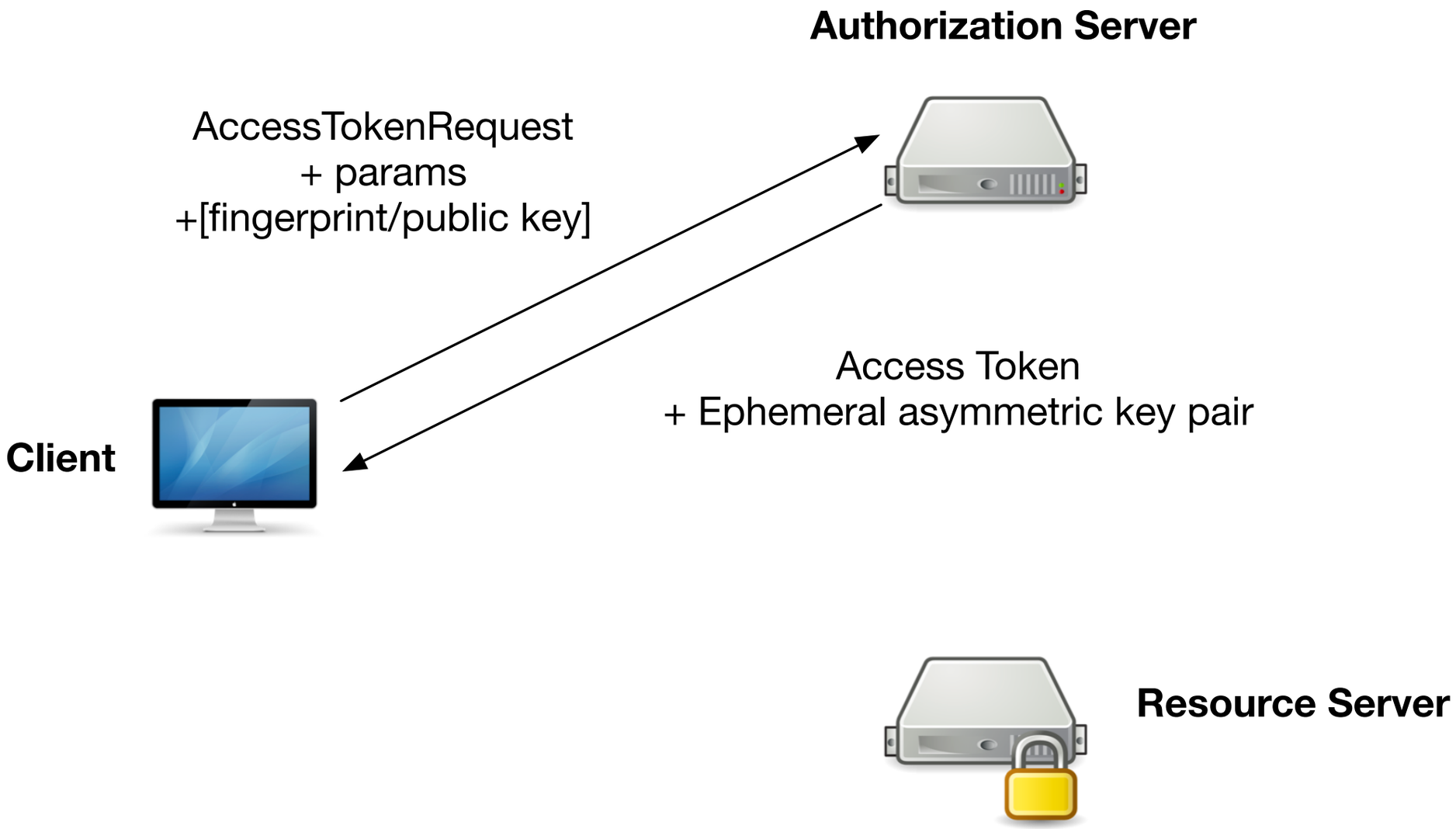
COMMUNICATING THE SESSION KEY AS->RS

(A) *by value* *AccessToken*

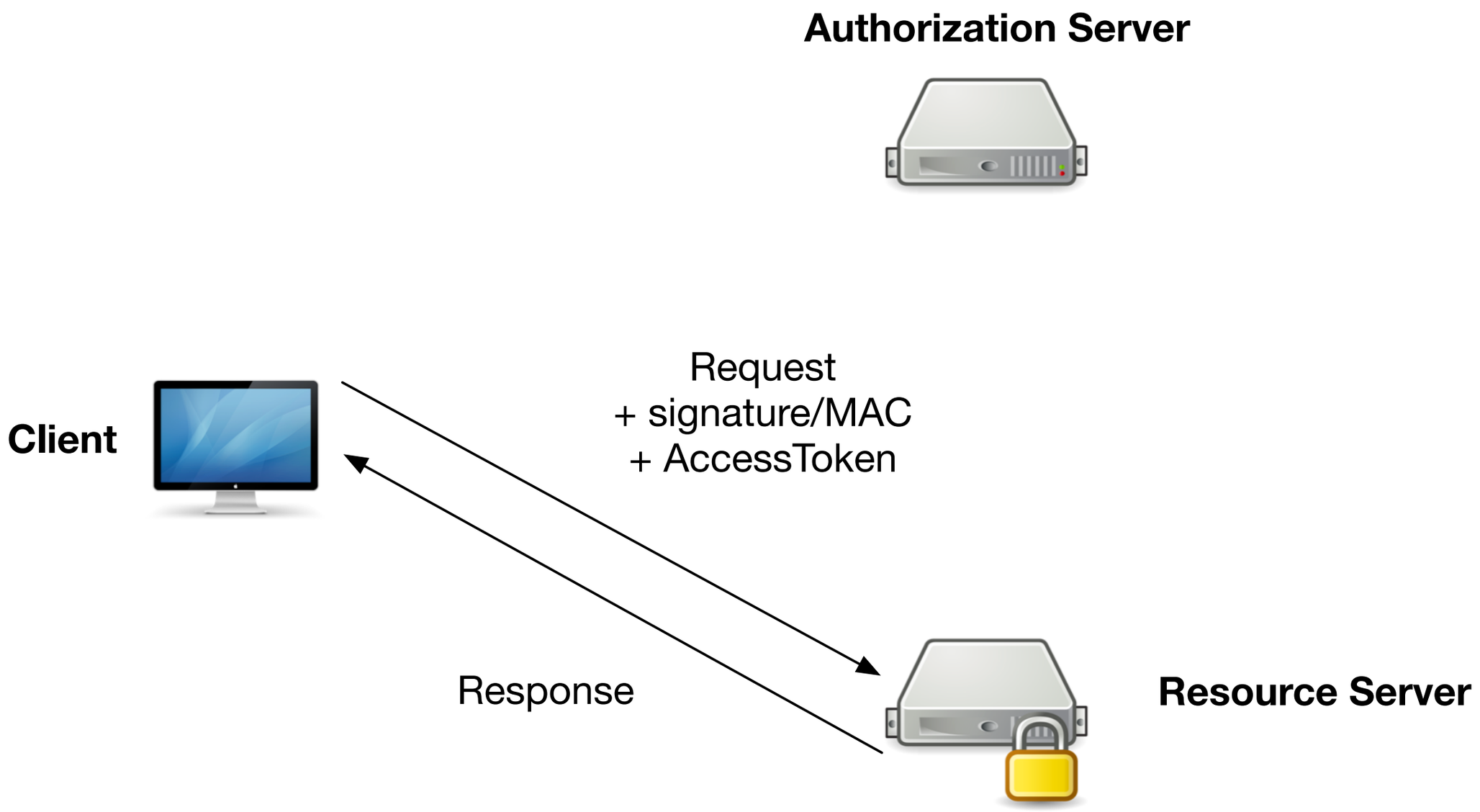
(B) *by reference* *Introspection*

(C) *Same back-end database*

[POP] CLIENT <-> AS INTERACTION (ASYM KEYS)

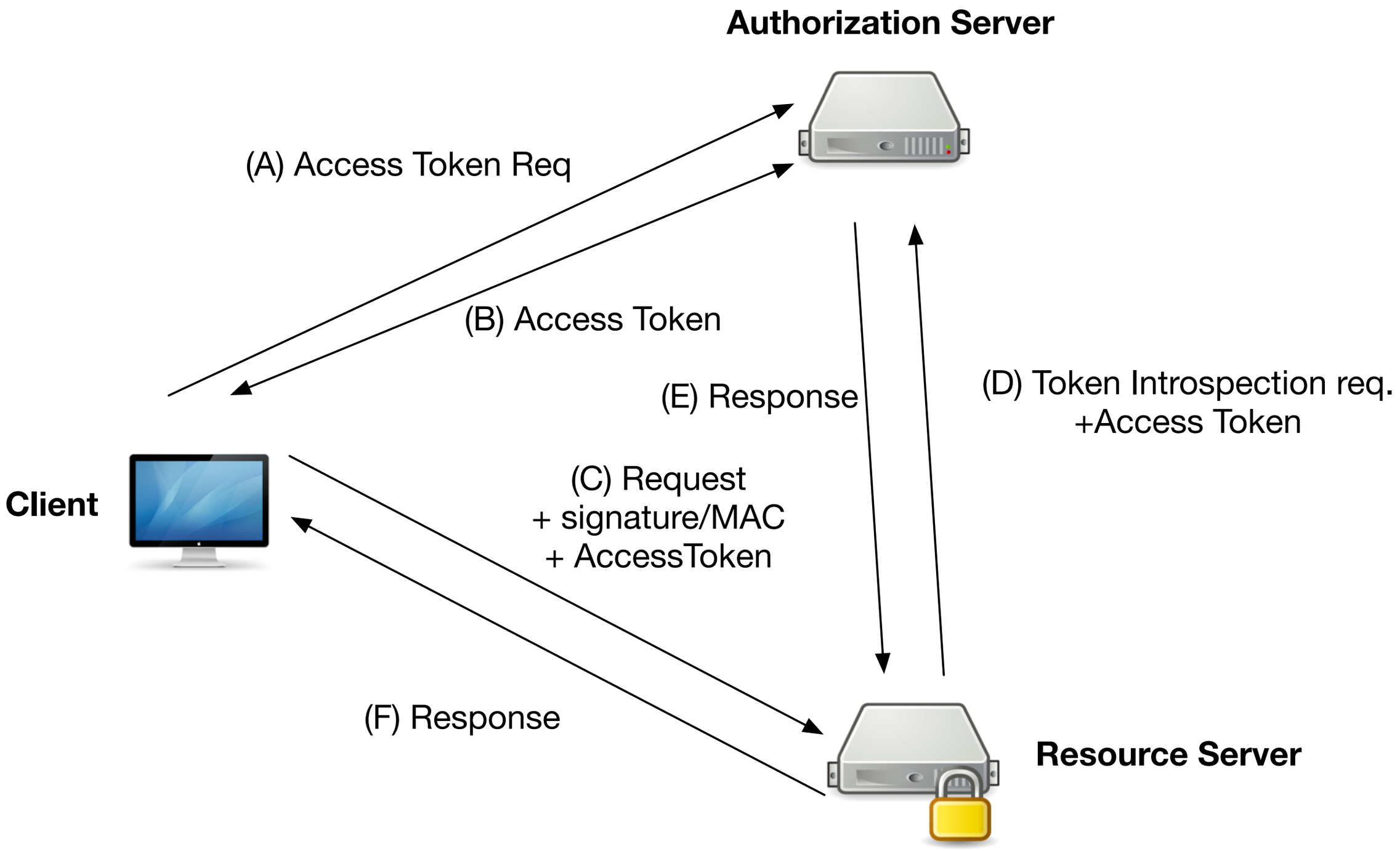


[POP] CLIENT AND RESOURCE SERVER INTERACTION



Tokens by value or by reference

[POP] RESOURCE AND AUTHORISATION SERVER INTERACTION



DRAFT-IETF-OAUTH-POP-KEY-DISTRIBUTION

- Extends the functionality of the token endpoint to allow keying material to be bound to an access token.
- The client can indicate which protected resource at what resource server it wants to access by using 'aud'.

CLIENT -> AS, SYMMETRIC KEY

POST /token HTTP/1.1

Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&alg=HS256

AS -> CLIENT SYMMETRIC KEY

HTTP/1.1 200 OK

Content-Type: application/json
Cache-Control: no-store


```
{
   "access_token":"SlAV32hkKG … (remainder of JWT omitted for brevity;
                                  JWT contains JWK in the cnf claim)",
    "token_type":"pop",
    "expires_in":3600,
    "refresh_token":"8xLOxBtZp8",
    "key" {
         "kty":"oct",
         "kid":"id123",
         "alg”:"HS256",
         "k":"ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE"
   }
}
```

CLIENT -> AS, ASYMMETRIC KEY

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

token_type=pop
&key=%7B%27kty%27%3A+%27RSA%27%2C+%27e%27%…
&redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
&code=SplxlOBeZQQYbYS6WxSbIA
&alg=RS256
&grant_type=authorization_code
```

```
{
  "kty": "RSA",
  "n": ”0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWh….”,
  "e": "AQAB",
  "alg": "RS256",
  "kid": ”id123"
}
```

AS -> CLIENT, ASYMMETRIC KEY

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache


```
{
        "access_token":"2YotnFZFE....jr1zCsicMWpAA",
        "token_type":"pop",
        "alg":"RS256",
        "expires_in":3600,
        "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

```
{
        "iss":"xas.example.com",
        "aud":"http://auth.example.com",
        "exp":"1361398824",
        "nbf":"1360189224",
        "cnf":{
          "jwk":{"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":"id123"
}}}
```

DRAFT-IETF-OAUTH-TOKEN-BINDING

- Cryptographically binds tokens to the TLS connection between the client and the token endpoint.
- Two use cases
  - First party (Refresh Tokens or Access Tokens)
  - Federation Use Cases