OAuth 2.0 & OpenID Connect — Cheat Sheet

RFC 6749 as amended by RFC 9700 · OpenID Connect Core 1.0 · OAuth 2.1 = **draft-ietf-oauth-v2-1-16** (not a standard)

ROLES

Resource owner	the user who grants access
Client	the app requesting the token
Authorization server	authenticates the user, issues tokens
Resource server	the API that accepts the token

*OAuth = delegated **authorization**. Authentication is OIDC.*

ENDPOINTS

`/authorize`	front channel, browser
`/token`	back channel, POST
`/introspect`	RFC 7662
`/revoke`	RFC 7009
`/userinfo`	OIDC
`/jwks`	public keys
`/.well-known/openid-configuration`	discovery

AUTHORIZATION REQUEST

```
GET /authorize?
 response_type=code
 &client_id=s6BhdRkqt3

&redirect_uri=https%3A%2F%2Fapp%2Fcb
 &scope=openid%20profile
 &state=af0ifjsldkj
 &code_challenge=E9Melhoa2O...
 &code_challenge_method=S256
```

`nonce`	OIDC replay defence
`prompt`	`none|login|consent`
`max_age`	force re-auth after N s
`acr_values`	ask for an assurance level
`response_mode`	`query|fragment|form_post`

TOKEN REQUEST & RESPONSE

```
POST /token
Authorization: Basic
czZCaGRSa3F0Mzo...
grant_type=authorization_code
&code=SplxlOBeZQ&redirect_uri=...
&code_verifier=dBjftJeZ4CVP...
```

```
{"access_token":"...","token_type":"
Bearer",

"expires_in":3600,"refresh_token":".
..",
 "scope":"openid
profile","id_token":"..."}
```

GRANT TYPES & 2.1 STATUS

grant_type	Spec	Status
`authorization_code`	6749+7636	**current**
`refresh_token`	6749 §6	**current**
`client_credentials`	6749 §4.4	**current**
`…:device_code`	8628	**current**
`…:token-exchange`	8693	**current**
`…:jwt-bearer`	7523	**current**
`…:saml2-bearer`	7522	**current**
`token` (implicit)	6749 §4.2	**REMOVED**
`password` (ROPC)	6749 §4.3	**REMOVED**

PKCE (RFC 7636)

```
verifier  = 43–128 char random
challenge =
BASE64URL(SHA256(verifier))
method    = S256          (never
"plain")
```

*Stops authorization-code interception. **2.1 requires it for every code client**, confidential ones included.*

state	CSRF on the redirect
nonce	ID-token replay
PKCE	code interception
iss	mix-up (RFC 9207)

TOKEN TYPES

	Audience	Purpose
Access	resource server	call the API
ID	the client	describe a login
Refresh	auth server	get a new access token

Never send an ID token to an API. Never treat an access token as proof of login.

REFRESH-TOKEN RULES

- **Rotate** on every use.
- **Reuse detection** → revoke the whole token family.
- Public clients: rotate *or* sender-constrain (RFC 9700 §4.14).
- Set both idle and absolute expiry.

JWT ANATOMY (RFC 7519)

```
header.payload.signature
(base64url)
{"alg":"RS256","kid":"k1","typ":"at+
jwt"}
{"iss":…,"sub":…,"aud":…,"exp":…,"ia
t":…,
 "scope":"read","cnf":{"jkt":"…"}}
```

JWS 7515 · JWE 7516 · JWK 7517 · JWA 7518 · BCP 8725 · `at+jwt` 9068

RESOURCE-SERVER CHECKLIST

- Reject **`alg:none`**; pin expected `alg`.
- Resolve `kid` against the JWKS only.
- Verify signature **before** reading claims.
- Check `iss`, `aud`, `exp`, `nbf` (allow small skew).
- Check `scope` / required claims.
- JWTs cannot be revoked before `exp`.

INTROSPECT / REVOKE

```
POST /introspect   token=…
&token_type_hint=access_token
→
{"active":true,"scope":"read","exp":
…}
```

```
POST /revoke       token=…
&token_type_hint=refresh_token
```

JWT = fast, no revocation. Opaque = revocable, one round trip. RFC 9701 returns a JWT.

CLIENT AUTHENTICATION

`client_secret_basic`	default
`client_secret_post`	body
`client_secret_jwt`	HMAC assertion
`private_key_jwt`	**7523, preferred**
`tls_client_auth`	**8705, mTLS**
`none`	public clients

DEVICE GRANT (RFC 8628)

```
POST /device_authorization →
device_code,
  user_code, verification_uri,
interval
POST /token grant_type=…:device_code
  → authorization_pending |
slow_down
    | access_denied | expired_token
```

Cross-device phishing: RFC 10027.

NATIVE APPS (RFC 8252)

- **MUST NOT use embedded WebViews.**
- Use the system browser or an **in-app browser tab**: `ASWebAuthenticationSession` (iOS), Custom Tabs (Android).
- Redirect: private-use scheme · claimed `https` · loopback.
- PKCE mandatory for public clients.

A real browser is what enables CAPTCHA, 2FA, passkeys and risk checks.

BROWSER APPS (RFC 10017)

- XSS defeats any token JS can read.
- **Prefer a BFF: token server-side, `HttpOnly+Secure+SameSite` cookie to the browser.**
- Browser-only: code+PKCE, rotation, DPoP.
- CSP as defence in depth.

SECURITY BCP (RFC 9700)

- Exact redirect-URI string matching.
- PKCE on every code flow.
- Validate `iss` (mix-up).
- One-time authorization codes.
- Rotate refresh tokens; detect reuse.
- No tokens in URLs, logs or `Referer`.
- TLS everywhere; verify certificates.
- No implicit, no ROPC.

SENDER-CONSTRAINED TOKENS

```
DPoP: proof JWT per request (RFC
9449)

{"htm":"POST","htu":"https://api/x",
  "iat":…, "jti":…, "ath":"…"}
 token carries cnf.jkt = key
thumbprint
```

mTLS (8705) binds via `cnf.x5t#S256`. Both re-add the proof of possession OAuth 1.0 had.

PAR · JAR · FAPI

PAR 9126	push the request, get `request_uri`
JAR 9101	signed/encrypted `request` object
FAPI 1.0/2.0	profiles that mandate them

OPENID CONNECT

```
scope=openid → id_token (a JWT)
{"iss","sub","aud","exp","iat","nonc
e",
 "auth_time","acr","amr","azp"}
```

UserInfo	`GET /userinfo` + bearer
Logout	RP-initiated · front · back channel
Discovery	`/.well-known/openid-configuration`

SCOPES VS. CLAIMS

- **scope** = capability the client asks for.
- **claim** = fact about the user/token.
- `resource` (8707) restricts the audience.
- **RAR** (9396) = structured, per-transaction authorization.

PASSWORDLESS *PRIMARY* METHODS

Method	amr	Phish-resist
E-mail OTP	`otp`	**no**
Magic link	`otp`	**no**
SMS OTP	`sms`	**no***
Push approval	`user`	**no**
Passkey	`swk`	**YES**
Device-bound	`hwk`	**yes**
Federated	upstream	inherits
Smart card	`hwk`	**yes**

** SP 800-63B: SMS is a **restricted** authenticator.*

2FA = AN ADDED *LAYER*

Never a primary method, never "passwordless" by itself.

Layer	amr
TOTP / HOTP app	`otp`
OTP as 2nd factor	`otp/sms`
Push as 2nd factor	`user`
Hardware token	`hwk`
Recovery codes	—

Combination	Passwordless?
password	**no**
password + TOTP	**no**
passkey	**yes**
passkey + OTT	**yes + MFA**

*`amr` containing `mfa` means **>1 factor**, not "no password". Check for the **absence of `pwd`**.*

STEP-UP (RFC 9470)

```
HTTP/1.1 401
WWW-Authenticate: Bearer

error="insufficient_user_authenticat
ion",
 acr_values="urn:…:aal2",
max_age=300
```

Client re-runs /authorize with `acr_values`; verify `acr`/`auth_time` on return.

ERROR CODES

`invalid_request`	malformed
`invalid_client`	client auth failed
`invalid_grant`	code/refresh bad, expired, reused, or redirect_uri mismatch
`unauthorized_client`	grant not allowed
`invalid_scope`	unknown scope
`access_denied`	user refused
`invalid_token`	401 at the RS
`insufficient_scope`	403 at the RS

RFC INDEX

6749/6750	core + bearer
5849	OAuth 1.0
7636	PKCE
7662 / 7009	introspect / revoke
7519 / 8725 / 9068	JWT / BCP / at+jwt
8252 / 10017	native / browser apps
8414 / 9728	AS / resource metadata
8628 / 8693	device / exchange
8705 / 9449	mTLS / DPoP
9126 / 9101	PAR / JAR
9207 / 9396 / 9470	iss / RAR / step-up
9700 / 6819	security BCP / threats

DISCOVERY DOCUMENT

```
issuer, authorization_endpoint,
token_endpoint,
jwks_uri, userinfo_endpoint,
end_session_endpoint,
scopes_supported,
response_types_supported,
grant_types_supported,
token_endpoint_auth_methods_supporte
d,
code_challenge_methods_supported,
dpop_signing_alg_values_supported
```

PROTOCOL → SPRING TYPE

client config	`ClientRegistration`
get a token	`OAuth2AuthorizedClientManager`
stored grant	`OAuth2AuthorizedClient`
validate JWT	`JwtDecoder` + `OAuth2TokenValidator`
introspection	`OpaqueTokenIntrospector`
scope	`SCOPE_` authority
ID token	`OidcUser`
customize claims	`OAuth2TokenCustomizer`
registered client	`RegisteredClient`
magic link	`oneTimeTokenLogin()`
passkeys	`webAuthn()`
MFA	`@EnableMultiFactorAuthentication`

*TOTP / SMS / push: **not built in**.*