OAuth 2.0 / OpenID Connect in Lightwave

# Introduction

Lightwave OIDC serves two main purposes.

1. Security Token Service (STS) in the sense that it issues security tokens in the form of JWT tokens (similar to WS-TRUST which issues SAML 2.0 Assertions)

2. single signon (SSO) / single logout (SLO) service for web-based applications (similar to SAML 2.0 Authentication Request and Single Logout protocols aka websso)

## Roles

User (Resource Owner)

User agent (browser)

Client (aka Relying Party / RP) e.g: lightwaveui

Authz Server (aka OpenID Provider / OP)

Resource Server (server that wants to implement authz for its API's) e.g: admin server

## JWT tokens

OAuth 2.0 defines an access\_token and a refresh\_token but does not specify their format. We are using JWT as the format.

OpenID Connect additionaly defines an id\_token and specifies its format as a JWT.

Lightwave OIDC issues id\_tokens, access\_tokens, and refresh\_tokens. All will be signed using the tenant's private key.

### id\_token

Represents the authentication of an end user at the Authz Server. The Client will validate this token and read its claims to sign in the end user.

example id\_token JSON body:

{

"sub": "administrator@lw-testdom.com",

"iss": "https://photon-63mciz57.lw-testdom.com/openidconnect/LW-TESTDOM.COM",

"groups": [

"LW-TESTDOM.COM\\Users",

"LW-TESTDOM.COM\\Administrators",

"LW-TESTDOM.COM\\CAAdmins",

"LW-TESTDOM.COM\\Everyone"

],

"token\_class": "id\_token",

"token\_type": "Bearer",

"given\_name": "Administrator",

"aud": "administrator@lw-testdom.com",

"scope": "at\_groups rs\_admin\_server openid offline\_access id\_groups",

"exp": 1501698283,

"iat": 1501694683,

"family\_name": "LW-TESTDOM.COM",

"jti": "rbdevuzSlBfwgSvQJdgtviOeVUmidOHcDZw79iy6ouQ",

"tenant": "LW-TESTDOM.COM"

}

### access\_token

The Client will use this in its protected resource requests against the Resource Server.

The Resource Server, on receiving the access\_token from the Client, will validate the token and make an API authz decision based on the user's group membership in the groups claim.

example access\_token JSON body:

{

"sub": "administrator@lw-testdom.com",

"aud": [

"administrator@lw-testdom.com",

"rs\_admin\_server"

],

"scope": "at\_groups rs\_admin\_server openid offline\_access id\_groups",

"iss": "https://photon-63mciz57.lw-testdom.com/openidconnect/LW-TESTDOM.COM",

"groups": [

"LW-TESTDOM.COM\\Users",

"LW-TESTDOM.COM\\Administrators",

"LW-TESTDOM.COM\\CAAdmins",

"LW-TESTDOM.COM\\Everyone"

],

"token\_class": "access\_token",

"token\_type": "Bearer",

"exp": 1501698283,

"iat": 1501694683,

"jti": "0lXE9y17uMevd0gBaIhRZ1ZqoYb7P64EuY7Bh7qyPXY",

"tenant": "LW-TESTDOM.COM",

"admin\_server\_role": "Administrator"

}

### refresh\_token

The Client uses this token to get a new access\_token (and id\_token) when the original access\_token expires (as reported by a Resource Server).

example refresh\_token JSON body:

{

"sub": "administrator@lw-testdom.com",

"aud": "administrator@lw-testdom.com",

"scope": "at\_groups rs\_admin\_server openid offline\_access id\_groups",

"iss": "https://photon-63mciz57.lw-testdom.com/openidconnect/LW-TESTDOM.COM",

"token\_class": "refresh\_token",

"token\_type": "Bearer",

"exp": 1501723483,

"iat": 1501694683,

"jti": "tKtIlifxWfgt8YpDZ0mwq8uF2xYxANlDnucHIOn7IAI",

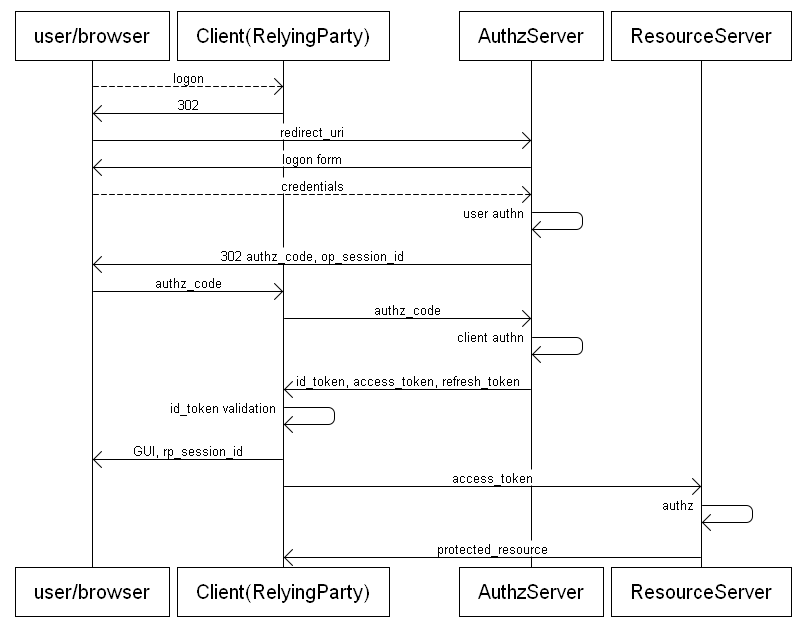
"tenant": "LW-TESTDOM.COM"

}

# Supported Flows

## Authz Code Flow

sequence diagram:



sample request/response:

Authn Request

GET /authorize?

response\_type=code&

client\_id=\_client\_id\_xyz\_&

redirect\_uri=https://client.example.com/cb&

state=\_state\_xyz\_&

nonce=\_nonce\_xyz\_&

scope=openid offline\_access HTTP/1.1

Host: server.example.com

Authn Response

HTTP/1.1 302 Found

Location: https://client.example.com/cb?

code=\_authz\_code\_xyz\_&

state=\_state\_xyz\_

Token Request

POST /token HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

grant\_type=authorization\_code&

code=\_authz\_code\_xyz\_&

redirect\_uri=https://client.example.com/cb&

client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&

client\_assertion=<private\_key\_jwt>

Token Response

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"<access\_token>",

"token\_type":"Bearer",

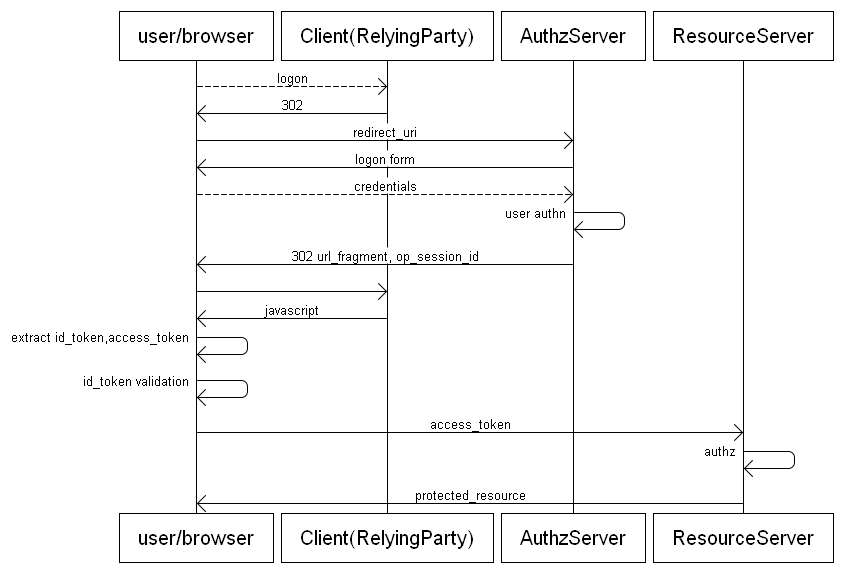
"expires\_in":3600,

"id\_token":"<id\_token>",

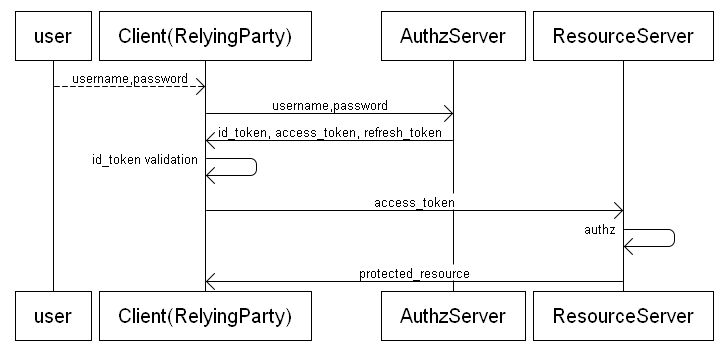
"refresh\_token":"\_refresh\_token\_xyz\_"

}

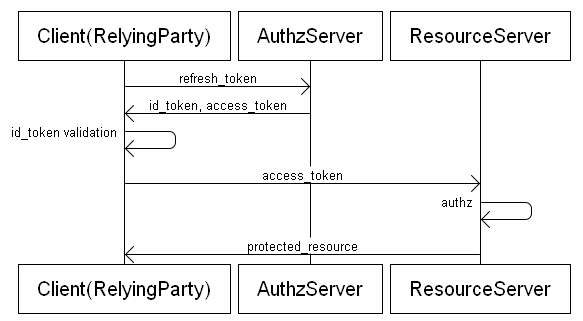
## Implicit Flow

1. sequence diagram:
2. 
3. sample request/response:
   1. Authn Request
   2. GET /authorize?
   3. response\_type=id\_token&
   4. client\_id=\_client\_id\_xyz\_&
   5. redirect\_uri=https://client.example.com/cb&
   6. state=\_state\_xyz\_&
   7. nonce=\_nonce\_xyz\_&
   8. scope=openid HTTP/1.1
   9. Host: server.example.com
   10. Authn Response
   11. HTTP/1.1 302 Found
   12. Location: http://example.com/cb#
   13. state=\_state\_xyz\_&
   14. id\_token=<id\_token>

## Password Flow

1. sequence diagram:
2. 
3. sample request/response:
   1. Token Request
   2. POST /token HTTP/1.1
   3. Host: server.example.com
   4. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
   5. Content-Type: application/x-www-form-urlencoded
   6. grant\_type=password&
   7. username=\_username\_xyz\_&
   8. password=\_password\_xyz\_&
   9. scope=openid offline\_access
   10. Token Response
   11. HTTP/1.1 200 OK
   12. Content-Type: application/json;charset=UTF-8
   13. Cache-Control: no-store
   14. Pragma: no-cache
   15. {
   16. "access\_token":"<access\_token>",
   17. "token\_type":"Bearer",
   18. "expires\_in":3600,
   19. "id\_token":"<id\_token>",
   20. "refresh\_token":"\_refresh\_token\_xyz\_"
   21. }

## Refresh Token Flow

1. sequence diagram:
2. 
3. sample request/response:
   1. Token Request
   2. POST /token HTTP/1.1
   3. Host: server.example.com
   4. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
   5. Content-Type: application/x-www-form-urlencoded
   6. grant\_type=refresh\_token&
   7. refresh\_token=\_refresh\_token\_xyz\_&
   8. client\_id=\_client\_id\_xyz\_
   9. Token Response
   10. HTTP/1.1 200 OK
   11. Content-Type: application/json;charset=UTF-8
   12. Cache-Control: no-store
   13. Pragma: no-cache
   14. {
   15. "access\_token":"<access\_token>",
   16. "token\_type":"Bearer",
   17. "expires\_in":3600,
   18. "id\_token":"<id\_token>"
   19. }

# Metadata endpoint

Spec link: <http://openid.net/specs/openid-connect-discovery-1_0.html>

OIDC Metadata endpoint publishes a JSON format document that includes server metadata such as server endpoints and server supported capability configurations. Client can use these data to help construct requests used in OIDC workflows.

Metadata endpoint format:

https://[OIDC Server URL]/{tenant}/.well-known/openid-configuration

Request to the metadata endpoint should use HTTP verb: GET. A sample request is listed below:

GET [OIDC Server URL]/{tenant}/.well-known/openid-configuration HTTP/1.1

Host: example.com

A successful response will return HTTP status code 200 (OK), and a JSON object using the application/json content type. The details for claim definitions can be found in the spec above. A sample response is listed below:

HTTP/1.1 200 OK

Content-Type: application/json

{

"response\_types\_supported":[

"code",

"id\_token",

"token id\_token"

],

"jwks\_uri":"https:\/\/photon-63mciz57.lw-testdom.com\/openidconnect\/jwks\/LW-TESTDOM.COM",

"end\_session\_endpoint":"https:\/\/photon-63mciz57.lw-testdom.com\/openidconnect\/logout\/LW-TESTDOM.COM",

"subject\_types\_supported":[

"public"

],

"id\_token\_signing\_alg\_values\_supported":[

"RS256"

],

"issuer":"https:\/\/photon-63mciz57.lw-testdom.com\/openidconnect\/LW-TESTDOM.COM",

"authorization\_endpoint":"https:\/\/photon-63mciz57.lw-testdom.com\/openidconnect\/oidc\/authorize\/LW-TESTDOM.COM",

"token\_endpoint":"https:\/\/photon-63mciz57.lw-testdom.com\/openidconnect\/token\/LW-TESTDOM.COM"

}

An error response will return HTTP status code 404 (Not Found) or 500 (Internal Server Error).

# JWKS endpoint

OIDC server also provides a JWK set endpoint to publish OIDC server public RSA keys in the format of JSON Web Key (JWK) set. The spec for JWK can be found in the link: <https://tools.ietf.org/html/draft-ietf-jose-json-web-key-41>.

Relying parties and resource servers need to retrieve this public key so they can verify token signature signed by the OIDC server.

JWK set endpoint format:

https://[OIDC Server URL]/jwks/{tenant}

Request to the JWK set endpoint should use HTTP verb: GET. A sample request is listed below:

GET [OIDC Server URL]/jwks HTTP/1.1

Host: example.com

A successful response will return HTTP status code 200 (OK), and a JSON object using the application/json content type. A sample response is listed below:

HTTP/1.1 200 OK

Content-Type: application/json

{

"keys":

[

{

"e": "AQAB",

"n": "......",

......

}

]

}

An error response will return HTTP status code 404 (Not Found) or 500 (Internal Server Error).

# Holder Of Key tokens

Solution Users or Clients that have a backing Solution User can request HOK tokens. The token request will include a proof of possession of the private key in the form of a solution\_user\_assertion or client\_assertion which are JWT's signed using the private key.

OIDC server will use the Solution User's registered certificate to validate the signature of the JWT.

The issued access\_token will contain an "hotk" claim that is a JWK Set containing the Solution User's public key.

The Client will then send signed requests to Resource Servers that support HoK access\_tokens.

The Resource Server will validate the signature of the request using the public key in the access\_token's "hotk" claim.

# Act As tokens

ActAs tokens are HOK tokens whose subject is a Person User. A token request that includes PersonUser and SolutionUser credentials at the same time will result in ActAs tokens.

There are many forms of Person User credentials, for example username/password or authorization code.

Solution User credentials will be in the form of solution\_user\_assertion or client\_assertion which are both JWT's signed using the Solution User's private key.

The most common form of acquiring ActAs tokens is the Authorization Code Flow. The authorization code represents Person User credentials (the user who logged) and the token request is required to have a client\_assertion which represents a registered OIDC client with a backing Solution User.

# Login

When a user is redirected to the Authz Server, if the Authz Server does not receive a session cookie, it will serve a login form.

The user can then login using username/password (or RSA SecurID, Kerberos, Smart Card, if enabled).

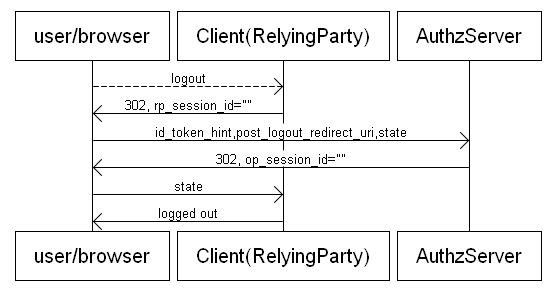
## Single Sign On

SSO is achieved by the use of cookies.

When a user logs in to the Authz Server (after being redirected to it by a Client), a session cookie is returned " oidc\_session\_id-<tenant>=<session\_id>".

When the user attempts to login to another Client, the other Client will redirect the user to the same Authz Server, the browser will then include the Authz Server session cookie which will allow the Authz Server to login the same user without prompting for credentials.

# Logout

1. Implemented according to [OpenID Connect Session Management 1.0 - draft 23](http://openid.net/specs/openid-connect-session-1_0.html) Section 5. "RP-Initiated Logout"
2. sequence diagram:
3. 
4. sample request/response:
   1. Logout Request
   2. GET /logout?
   3. id\_token\_hint=<id\_token>&
   4. post\_logout\_redirect\_uri=https://client.example.com/cb&
   5. state=\_state\_xyz\_
   6. Host: server.example.com
   7. Cookie:oidc\_session\_id-vsphere.local=<session\_id>
   8. Logout Response
   9. HTTP/1.1 302 Found
   10. Location: https://client.example.com/cb?
   11. state=\_state\_xyz\_
   12. Set-Cookie:oidc\_session\_id-vsphere.local=""

## Single Log Out

* 1. Implemented according to [OpenID Connect HTTP-Based Logout 1.0 - draft 00](http://openid.net/specs/openid-connect-logout-1_0.html)
  2. When Authz Server receives a logout request, it returns an html page with iframe tags whose source is the logout\_uri of all the Clients the user has logged into.
  3. The browser will make HTTP GET requests and include session cookies which will allow the Client to logout the user.

# Implementation

## Server State

The following classes keep track of server state using LinkedHashMaps (wrapped by oidc.server.SlidingWindowMap):

|  |  |  |  |
| --- | --- | --- | --- |
| class | purpose | map key | map value |
| SessionManager | HTTP session cookie maps to a user logged in to a number of Clients | SessionID | PesonUser, LoginMethod, Set<ClientID>, Set<ClientInfo> |
| AuthorizationCodeManager | state for issued authz codes | AuthorizationCode | PersonUser, SessionID, original AuthenticaitonRequest |

## Request Processing

Every endpoint has a Controller which is the first point of processing:

MetadataController

JWKSController

AuthenticationController

TokenController

LogoutController

Metdata and jwks requests are simple GET requests and will be handled directly by their corresponding Controller whereas the rest will be handled by RequestProcessor's:

AuthenticationRequestProcessor

TokenRequestProcessor

LogoutRequestProcessor

These will encapsulate requests and responses:

AuthenticationRequest

AuthenticationResponse

AuthenticationSuccessResponse

AuthenticationErrorResponse

TokenRequest

TokenResponse

TokenSuccessResponse

TokenErrorResponse

LogoutRequest

LogoutResponse

LogoutSuccessResponse

LogoutErrorResponse

## Code Organization

Code is organized into 5 components that build into separate jars

|  |  |  |
| --- | --- | --- |
| component | role | dependencies |
| server | request processing, state management | common, protocol |
| client | client library | common, protocol |
| protocol | request and response encapsulation, token definition | common |
| common | public types and enums |  |
| sample | sample Client/Relying Party, builds into a war file | common, client |

# Data Sheet

## Endpoints

|  |  |  |
| --- | --- | --- |
| endpoint | supported? | usage |
| authorization\_endpoint | ✓ | interactive user login |
| token\_endpoint | ✓ | obtain id\_token,access\_token,refresh\_token |
| userinfo\_endpoint |  | obtain user profile information |
| jwks\_uri | ✓ | obtain public key corresponding to the private signing key (per-tenant) |
| discovery | ✓ | OIDC Discovery |
| registration\_endpoint |  | OIDC Dynamic Client Registration |
| end\_session\_endoint | ✓ | logout |
| check\_session\_iframe |  | a way of implementing single logout |

## Response types

|  |  |  |
| --- | --- | --- |
| response\_type | supported? | usage |
| code | ✓ | authz code flow |
| id\_token | ✓ | implicit flow |
| id\_token token | ✓ | implicit flow |
| token |  | implicit flow (OAuth 2.0) |
| code id\_token |  | hybrid flow |
| code id\_token token |  | hybrid flow |
| code token |  | hybrid flow |

## Grant types

|  |  |  |
| --- | --- | --- |
| grant\_type | supported? | usage |
| authorization\_code | ✓ | authz code flow |
| implicit | ✓ | implicit flow |
| password | ✓ | password flow |
| refresh\_token | ✓ | refresh token flow |
| certificate | ✓ | extension grant type |
| gss ticket | ✓ | extension grant type |
| securid | ✓ | extension grant type |
| perosn user certificate | ✓ | extension grant type |
| client\_credentials | ✓ | client credentials flow |
| JWT |  | JWT Profile for Authz Grants |
| SAML 2.0 |  | SAML 2.0 Profile for Authz Grants |

## Subject identifier types

|  |  |
| --- | --- |
| subject\_type | supported? |
| public | ✓ |
| pairwise |  |

## Client authentication methods

|  |  |
| --- | --- |
| token\_endpoint\_auth\_method | supported? |
| client\_secret\_basic |  |
| client\_secret\_post |  |
| client\_secret\_jwt |  |
| private\_key\_jwt | ✓ |

## JWT signature algorithms

|  |  |
| --- | --- |
| signature algorithm | supported? |
| RS256 | ✓ |
| RS384 |  |
| RS512 |  |
| PS256 |  |
| PS384 |  |
| PS512 |  |

## Error Codes

|  |  |
| --- | --- |
| OAuth2 .0 error code | returned? |
|  |  |
| invalid\_request | ✓ |
| unauthorized\_client | ✓ |
| invalid\_client | ✓ |
| invalid\_scope | ✓ |
| unsupported\_response\_type | ✓ |
| unsupported\_grant\_type | ✓ |
| invalid\_grant | ✓ |
| access\_denied | ✓ |
| server\_error | ✓ |
| temporarily\_unavailable |  |
|  |  |
| OpenID Connect error code |  |
|  |  |
| interaction\_required |  |
| login\_required |  |
| account\_selection\_required |  |
| consent\_required |  |
| invalid\_request\_uri |  |
| invalid\_request\_object |  |
| request\_not\_supported |  |
| request\_uri\_not\_supported |  |
| registration\_not\_supported |  |