DRS Platform

Interface Security & Authentication

V0.1 Initial draft

29th January 2023

Document Control

Version	Date	Author(s)	Rationale
0.1	29/01/2023	James Denning james.denning.ext@rev-log.com	Initial vertsion

1. INTRODUCTION

The interfaces used for integration require securing to prevent unauthorised use. In summary that security will consist of:

1. A minimum level of **encryption** – this will be centred around TLS v1.2 using x509 certificates that are issued with a chain of trust to a trusted certificate authority that is generally acceptable to well-known operating systems and browsers
2. **Authentication** – an OAuth 2.0 client credential flow where an integration partner is provided with credentials (a client id and client secret) with which they obtain a bearer token that is then attached to API requests.

This specification will be adhered to by all interfaces hosted by RLG within the CSL solution and is asserted as the required minimum standard for all integration partners unless by explicit agreement.

The standards are based on long-established well-known industry standards and adherence should be feasible for reasons up to date technology stacks, programming languages and toolsets. It is usually simply within the terminology used that there is variance that can lead to uncertainty – please liaise with the RLG CLS programme technical team for further clarification.

2. ENCRYPTION

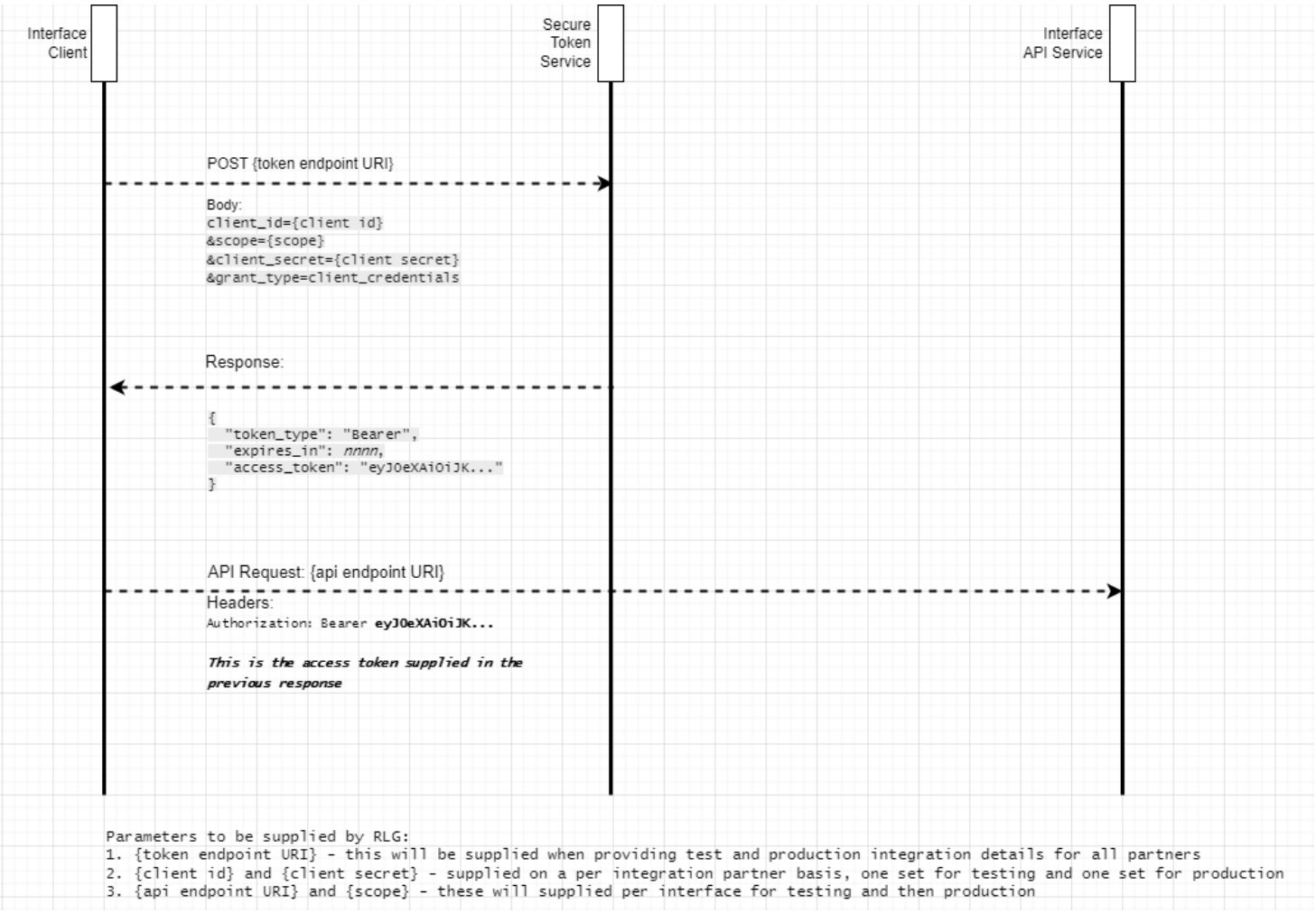
Core integration APIs for the RLG CLS solution are designed to use the HTTP 1.2 specification as laid out in RFC 2616 (https://www.rfc-editor.org/rfc/rfc2616). The transport layer they should operate on should be encrypted to the TLS 1.2 standard (https://www.ietf.org/rfc/rfc5246.txt) using one of the widely accepted list of cipher suites (as listed in the Appendices) as established during initial connection handshake.

- The certificates will be valid date-wise
- Common Name will match the host name being used (either through single name assignment to that explicit host or via wildcard).
- Therefore no errors or software security overrides should be necessary to communicate with an API endpoint.
- No private key need be shared between parties.

What this really means is that the URL will begin with https and http client software and tools on most common reasonably up to date operating systems/stacks should have no problems establishing a secure encrypted session with the remote server endpoint just like putting a secure internet address in a browser address bar with no errors or warnings.

3. AUTHENTICATION

The OAUTH2 Client Credentials flow as defined by RFC 6749 section 4.4 (https://tools.ietf.org/html/rfc6749#section-4.4) – use a “client_id” and “client_secret” as supplied by RLG/CSL to authenticate to a Secure Token Service to receive an access_token that is then supplied in the Authorization header of subsequent requests to the API.



- Token expiry: this is specified in seconds in the response body. A client should reuse the token within its lifetime and not retrieve a new token for every API request it needs to make. Token expiry is set to between 60 and 90 minutes (with ~75 minutes being the average).
- Refresh tokens: not provided/supported
- Tokens should be treated as secrets.

Guidance notes:

1. The actual implementation will use Microsoft Azure AD – further details can be found at https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-client-creds-grant-flow - note the client consent will already be granted and we will supply tenant id.
2. Most common programming languages/platforms have http client libraries/extensions that make obtaining the token easy – examples can be found at https://learn.microsoft.com/en-us/azure/active-directory/develop/sample-v2-code#service--daemon

4. APPENDICES

4.1 Cipher Suites

The following Cipher Suites are those that will be used – as stated this should NOT be a problem for most reasonably up to date operating systems/stacks but for the purpose of clarity they are specified here. Note cipher suites in **bold** are the preferred suites, those in *italics* will be supported but have a finite lifetime due to obsolescence.

- **TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384**
- **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256**
- **TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384**
- **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256**
- *TLS_DHE_RSA_WITH_AES_256_GCM_SHA384*
- *TLS_DHE_RSA_WITH_AES_128_GCM_SHA256*