

Inland Revenue

## Identity and Access Services build pack

**Date:** 24//11/20233

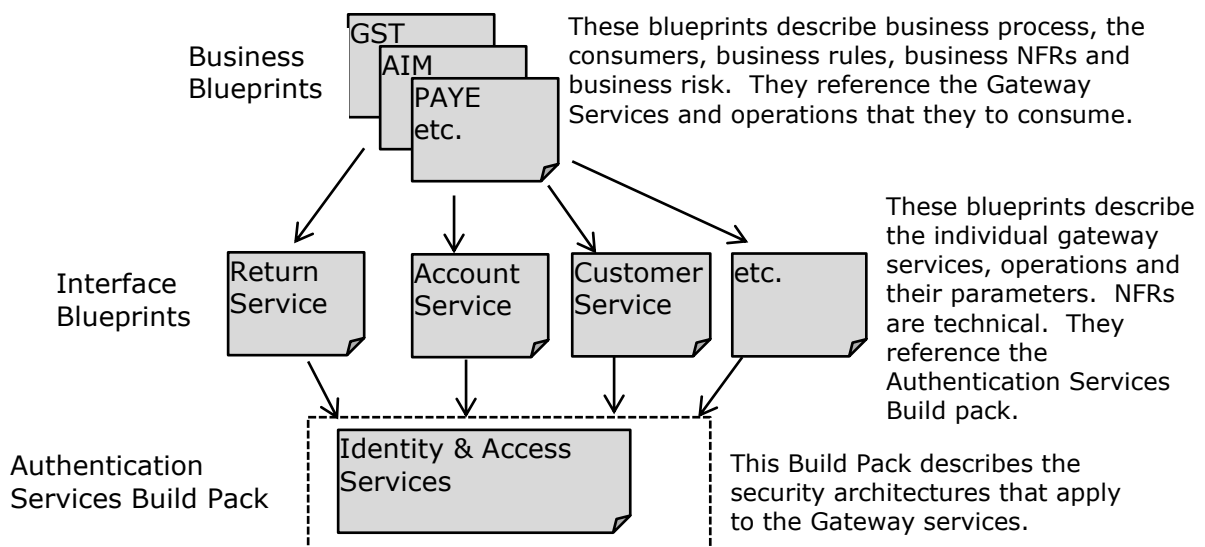
UNCLASSIFIED

## About this Document

This document is intended to provide Digital Service Providers with the technical detail required to consume the Identity and Access services offered by Inland Revenue.

This is a technical document that supports the on boarding processes of an end-to-end solution. The associated on-boarding document(s) describe the end-to-end business level solution, of which this build pack is part. This document describes the architecture of the technical solution, the interaction with other build packs, schemas and endpoints. Also included are sample payloads to use in non-production environments.

It is part of a 3-tier hierarchy of documents that are depicted in the following diagram:



## Contents

<b>1 Overview.....</b>	<b>5</b>
1.1 This solution .....	5
1.1.1 Organisational Authentication and Authorisation. ....	6
1.1.2 End-User Authentication and Authorisation. ....	6
1.2 Intended audience.....	6
1.3 Information IR will provide Service Providers .....	7
1.3.1 Token Auth (Cloud or Native) .....	7
1.3.2 SSH keys .....	7
1.4 Information Service Providers must provide IR .....	7
1.4.1 Service Provider Information .....	7
1.4.2 Token Auth (Cloud or Native) .....	7
1.4.3 SSH keys .....	7
<b>2 Description of the IR Authentication Mechanisms.....</b>	<b>8</b>
2.1 IR Token Auth Implementation using OAuth 2.0 .....	8
2.1.1 High Level View of OAuth 2.0 .....	8
2.1.2 OAuth Services .....	9
2.1.3 Authorisation Service.....	9
2.1.4 Refresh Token Service .....	17
2.1.5 Introspection Token Service (Validate).....	18
2.1.6 Revocation Token Service (Revoke).....	20
2.1.7 Security Considerations .....	21
2.1.8 PKCE Support .....	21
2.1.9 Endpoints.....	22
<b>2.2 Native Application Token Auth .....</b>	<b>23</b>
<b>2.3 SSH Keys.....</b>	<b>24</b>
<b>2.4 M2M using Client Signed JWT .....</b>	<b>24</b>
<b>3 TLS &amp; Mutual TLS Authentication .....</b>	<b>29</b>
3.1 Ciphers .....	29
3.2 Mutual TLS Authentication (mTLS).....	29
<b>4 Appendix A – Sample payloads .....</b>	<b>30</b>
4.1 Request Authorisation Code.....	30
4.1.1 Request .....	30
4.2 Authorisation Code response .....	30
4.2.1 Success Response – Authorisation Code sent .....	30
4.3 Request Access token .....	30
4.3.1 Exchange Authorisation Code for OAuth Access Token. ....	30
4.3.2 Success Response – Access Token sent.....	31
4.4 Request Refresh token .....	31
4.4.1 Refresh request .....	31

---

4.4.2	Refresh token reply .....	31
4.4.3	Error Response. ....	32
4.5	Introspection token request (Validate) .....	32
4.5.1	Introspection token request.....	32
4.5.2	Introspection token reply (Validate) .....	33
4.5.3	Introspection token Invalid Token Response. ....	33
4.6	Revocation token request (Revoke) .....	33
4.6.1	Revocation token request (Revoke) .....	33
<b>5</b>	<b>Appendix B – Glossary .....</b>	<b>34</b>
<b>6</b>	<b>Appendix C—Creating a signing certificate and public/private key pair ..</b>	<b>36</b>
<b>7</b>	<b>Appendix D – Deprecated, Aged and Changing Standards.....</b>	<b>38</b>
<b>8</b>	<b>Appendix E—Change log.....</b>	<b>39</b>

## 1 Overview

### 1.1 This solution

Inland Revenue (IR) is establishing a new set of Identity and Access services. These will provide Service Providers with authentication and authorisation mechanisms for accessing IR's new Gateway Services.

The specifications within this document refer to specific cryptographic standards (e.g. security protocols, key lengths, signing and hashing algorithms etc) that are applicable at the time of writing. IR reserves the right to upgrade these in response to factors such as external threats, industry standards and changes to NZISM. IR's partners will be consulted, via IR's relationship managers, about upgrades to these standards.

There are two distinct types of entity for which IR provides mechanisms for authentication and authorisation:

1. Organisations
2. End Users

The mechanisms are as per the diagram below:

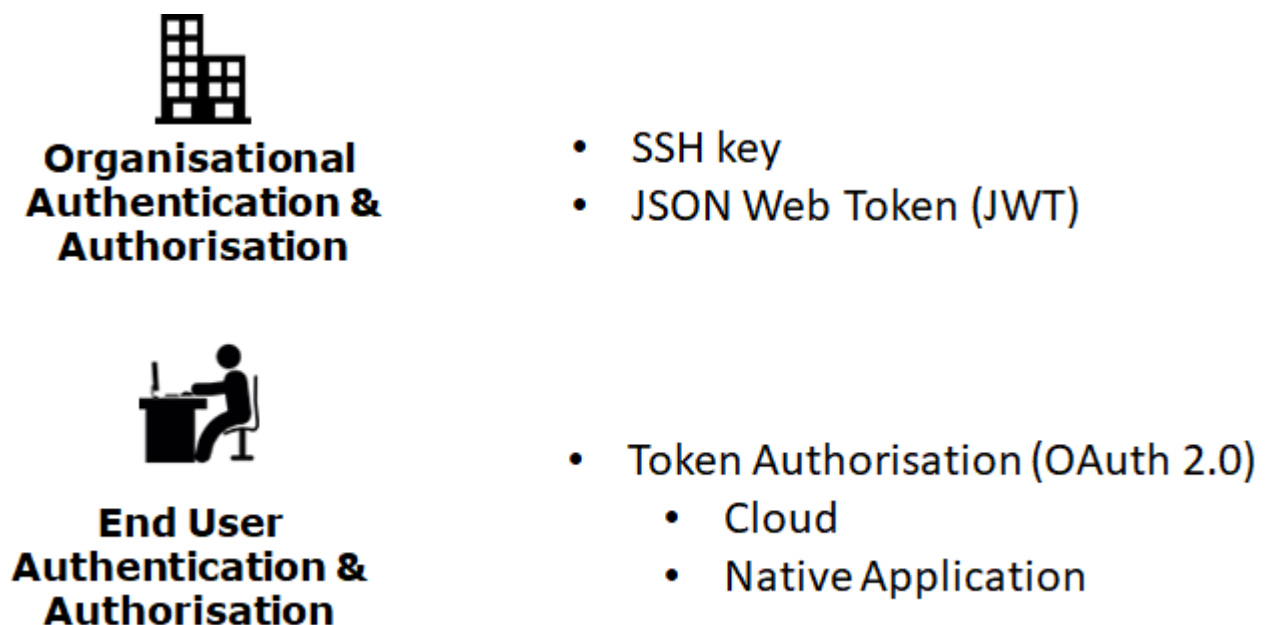


Figure 1 Types of Entity and Authentication & Authorisation mechanisms

### 1.1.1 Organisational Authentication and Authorisation.

The table below details the current and future mechanisms IR provide to authenticate and authorise an organisation for Machine to Machine (M2M) communication.

Authorisation Mechanism	Uses
<b>SSH keys</b>	This mechanism is used in SFTP file transfers to identify the organisations sending/receiving files.  SSH keys need to be exchanged to authenticate both parties.
<b>JSON Web Token (JWT) Using JSON Web Signature (JWS) for digital signatures with RSA/ECDSA</b>	Used to sign messages in order to identify the service provider or a customer of a service provider to IR.  This will be used by IR as a M2M mechanism.

**Table 1 Organisational Authentication and Authorisation Methods**

### 1.1.2 End-User Authentication and Authorisation.

The OAuth 2.0 process is used to authenticate end-users using their IR user ID and password and grant 3<sup>rd</sup> party software consent to access their information.

The OAuth 2.0 mechanism to be used by a service provider is based upon the nature of the client application the end-user will be using.

Authorisation Mechanism	Uses
<b>Cloud application Token Auth OAuth2.0 based</b>	Use when the client application is a web-enabled cloud-based application. It requires an online user to enter their myIR user ID and password to grant the application access to their IR information.
<b>Native application Token Auth OAuth2.0 based</b>	Use when the client application is a desktop or other native application. It also requires an online user to enter their myIR user ID and password to grant the application access to their IR information.  See section 2.2 Native Application Token Auth below for details

**Table 2: End-User Authentication and Authorisation Methods**

## 1.2 Intended audience

This build pack and the resources to which it refers are primarily focused on the needs of Software Developers' technical teams and development staff.

The reader is assumed to have a suitable level of technical knowledge in order to comprehend the information provided. A range of technical terms and abbreviations are used throughout this document, and while most of these will be understood by the intended readers, a glossary is provided at the end.

This document is not intended for use by managerial staff or those with a purely business focus.

### **1.3 Information IR will provide Service Providers**

#### **1.3.1 Token Auth (Cloud or Native)**

1. URLs and parameters for invoking the Authentication services.
2. Client ID (agreed with service consumer)
3. Client secret (used in step 2 in section 2.1.2)

#### **1.3.2 SSH keys**

1. SSH keys for SFTP.
2. PGP public keys if used for payload encryption and signing

### **1.4 Information Service Providers must provide IR**

#### **1.4.1 Service Provider Information**

1. Full business name
2. Client ID (agreed with IR and used in requests to IR))
3. Key Contact
4. Email of key contact or delegate
5. Mobile Phone number (SMS may be used for some information)
6. IP addresses Service Providers will use for test instances for IR firewall whitelisting.

#### **1.4.2 Token Auth (Cloud or Native)**

1. Redirect URI for Authorisation code and Authentication token.

#### **1.4.3 SSH keys**

1. SSH public keys if using SFTP.
2. Their own Public Keys if using PGP.

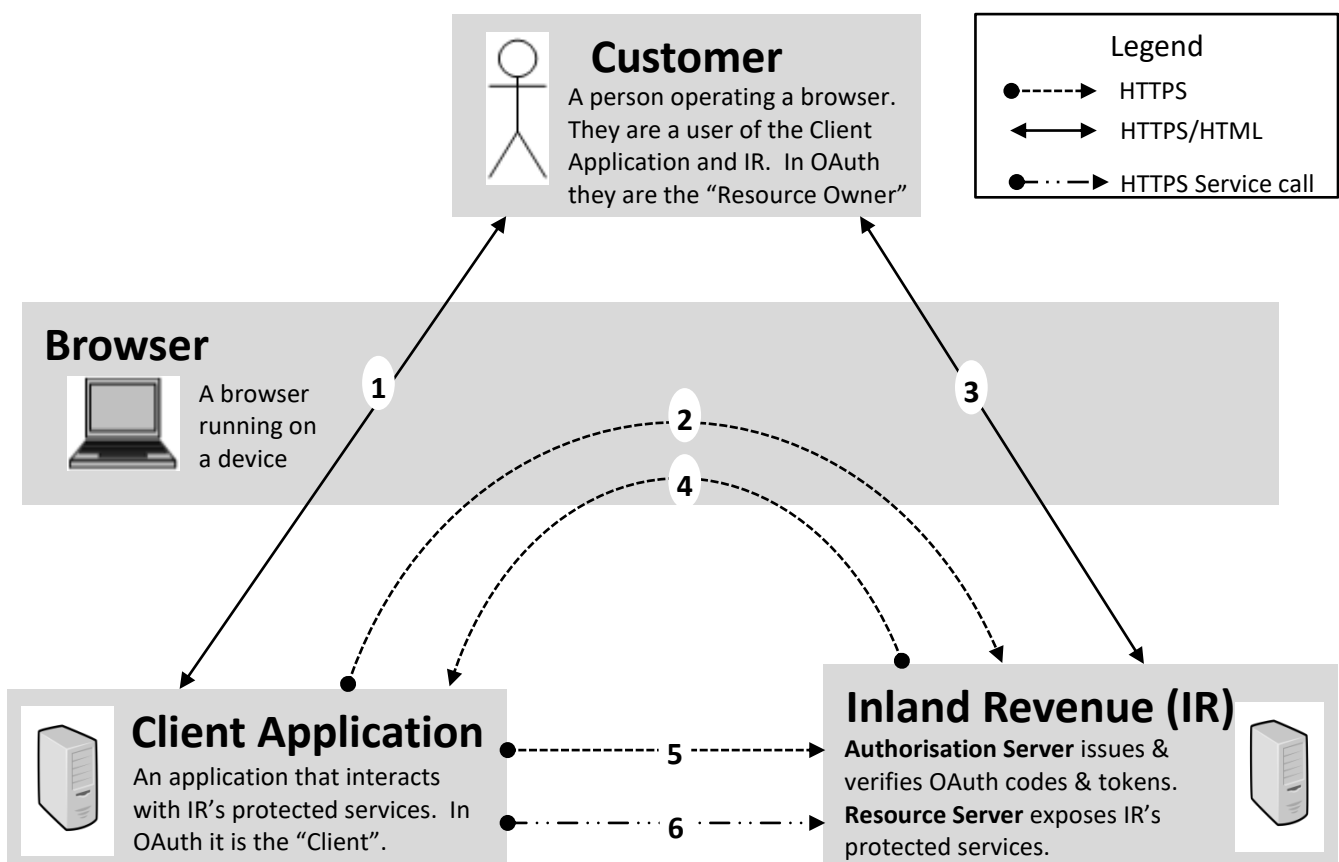
## 2 Description of the IR Authentication Mechanisms

### 2.1 IR Token Auth Implementation using OAuth 2.0

This section describes the IR OAuth 2.0 implementation. This high-level description covers the services offered by IR's implementation of OAuth Authorisation services including both Cloud and Native application usage. This version of this document (v3.0) only details the behaviour of START IAM. For previous version the Oracle XIAMS please refer to the archived version of this build pack on [GitHub](#).

#### 2.1.1 High Level View of OAuth 2.0

For OAuth 2 the following diagram depicts the high-level end-to-end view of the components and the interactions between them:



1. The User is interacting with the Client Application. They access a protected service provided by IR (e.g. to file a return, retrieve a balance etc.)
2. The Client Application invokes the Authorisation API to get an authorisation code, the user's browser is redirected to IR's logon page.
3. IR prompts the User to logon, they are authenticated. On first use the User must also supply their consent for the Client Application to access IR on their behalf. IR issues the Authorisation Code.
4. The Authorisation Code is returned to the Client Application (via the browser).
5. The Client Application invokes IR's Token service to redeem the Authorisation Code for an OAuth Access Token. It has a finite time to live.
6. The Client Application can then invoke IR's protected services (e.g. to file a return etc.) supplying the OAuth Access Token in the header. The OAuth Access Token can be used for multiple invocations until it expires.



Inland Revenue's implementation of the OAuth 2 standard conforms to the Authorisation Code Grant flow described in section 4.1 of RFC 6749 ( <https://tools.ietf.org/html/rfc6749> ).

### 2.1.2 OAuth Services

This section describes the services offered by IR through the Authorisation services, including:

- a) Authorisation Service (via OAuth 2.0)
- b) Token Service
- c) Introspection Token Service (Validate Token)
- d) Revocation Token Service (Revoke Token)

Not including the Authorisation Service, query parameters must be added in the POST body. Authorisation Service query parameters in the POST request will not be accepted.

The following sections will describe the steps and service calls required to integrate with the IR implementation.

### 2.1.3 Authorisation Service

This section describes the steps and service calls required when using the IR implementation of OAuth 2.0. These are the same for both Cloud and Native App usage.

#### 2.1.3.1 Customer accesses the Client Application (Step1)

The Customer accesses the Client application and triggers the need for it to consume one of Inland Revenue's protected services (e.g. to retrieve an account balance, to file a return etc.).

#### 2.1.3.2 Request Authorisation Code (Step 2)

The customer's browser is redirected to the IR Authorisation service to authenticate the user and confirm scope using the GET method described below (example is for one IR Test environment – see 2.1.9 below for all endpoints):

```
https://test5.services.ird.govt.nz/gateway3/oauth/authorize?
response_type=code

&client_id=IdOfCompanyUsingTheAPI

&redirect_uri=http://client.example.com/return

&scope=MYIR.Services

&state=xyz
```

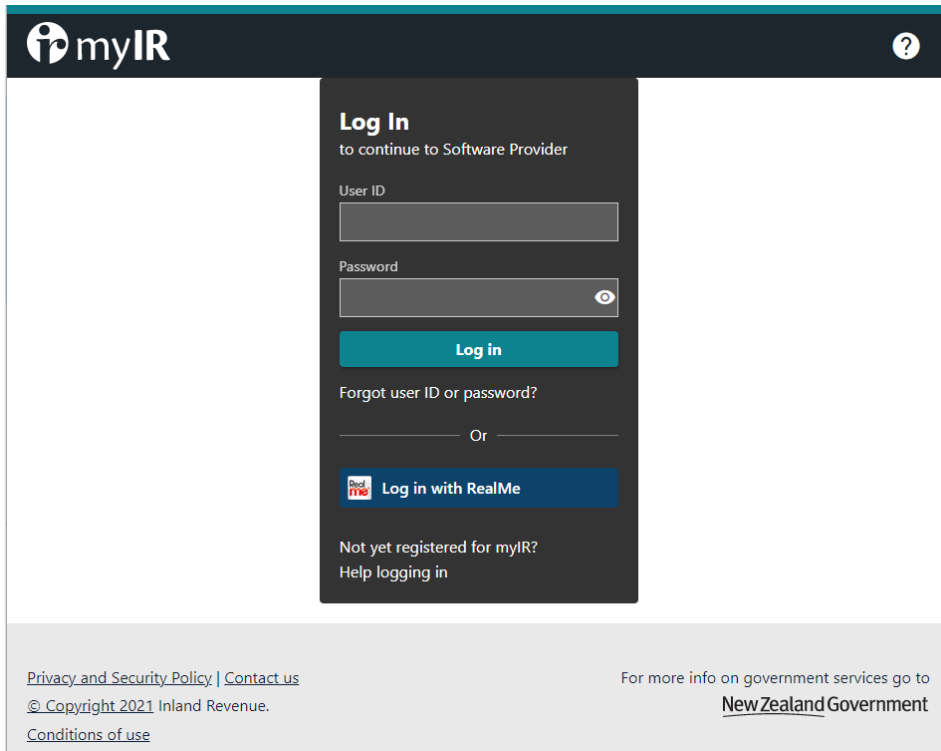
Name	Description	Required	Valid Values
<b>response_type</b>	Response type requested. Indicates that your server expects to receive an authorization code.	Required	"code"

<b>client_id</b>	The agreed Client identifier established at registration. Inland Revenue maintains this list of values.	Required	client_id
<b>redirect_uri</b>	The Client application's redirect URI to which the Authorisation Code is returned.	Required	Business Partner defined & agreed with IR
<b>scope</b>	Use space-separated values. Define scope values in the configuration/scope registry.	Required	"MYIR.Services"
<b>state</b>	A value used by the Business Partner to maintain state between the request and callback. The use of the state parameter is strongly recommended for additional security. The value should relate to the logged-in user, and allows the Client (software provider) to confirm that the request being received at their Redirect URI is valid. See <a href="https://tools.ietf.org/html/rfc6749">https://tools.ietf.org/html/rfc6749</a> for more information (Section 12.12).	Optional, but recommended	Business Partner defined.  Must be < 200 characters.  Allowed Characters: a-z A-Z 0-9 - . ? , : / \ + = \$ #  Not space (%20, (ASCII 32))
<b>PKCE Support – See section <a href="#">2.1.8</a> about PKCE.</b>			
<b>code_challenge</b>	SHA256 hash of the Code Verifier.	Optional	BASE64 URL encoded string value of the hash Code-verifier.
<b>code_challenge_method</b>	Hashing. Algorithm used to hash the Code Verifier.	Optional	Method used to generate the code_challenge. Must be S256.

### 2.1.3.3 *Login with myIR Credentials (step 3)*

During this step the customer will be required to authenticate and will be redirected to the following myIR logon screen. For OAuth 2.0 for Native Apps this authorisation request is in an external user-agent (typically the browser).

The authorisation page has 'iframe detection' and will prevent the authorisation page from loading within an iframe.



The image shows the myIR Log In screen. At the top is the myIR logo and a help icon. The main heading is "Log In" with the subtext "to continue to Software Provider". Below this are input fields for "User ID" and "Password" (with a toggle for visibility). A teal "Log in" button is present. Below the button is a link for "Forgot user ID or password?". An "Or" separator is followed by a "Log in with RealMe" button. At the bottom, there are links for "Not yet registered for myIR?" and "Help logging in". The footer contains links for "Privacy and Security Policy", "Contact us", "Conditions of use", and a note about government services from the New Zealand Government.

**Note:** The customer must already have an IR Online Services credential, i.e., a myIR logon.

Example of invalid redirect URI, HTTP: 400 response code.

```
{
  "error": "invalid_request",
  "error_description": "Invalid redirect_uri. Provided redirect_uri
(http://localhost/web services/gateway2/SSC/SignIn) is not configured for
this client."
}
```

Example of invalid clientID, HTTP: 401 response code.

```
{
  "error": "invalid_client",
  "error_description": "Client is invalid."
}
```

Example of invalid\_scope will redirect the return\_uri

```
https://client.example.com/return?error=invalid_scope&error_description=Invalid+scope+requested&state=xyz
```

Example of missing response\_type, HTTP: 400 response code.

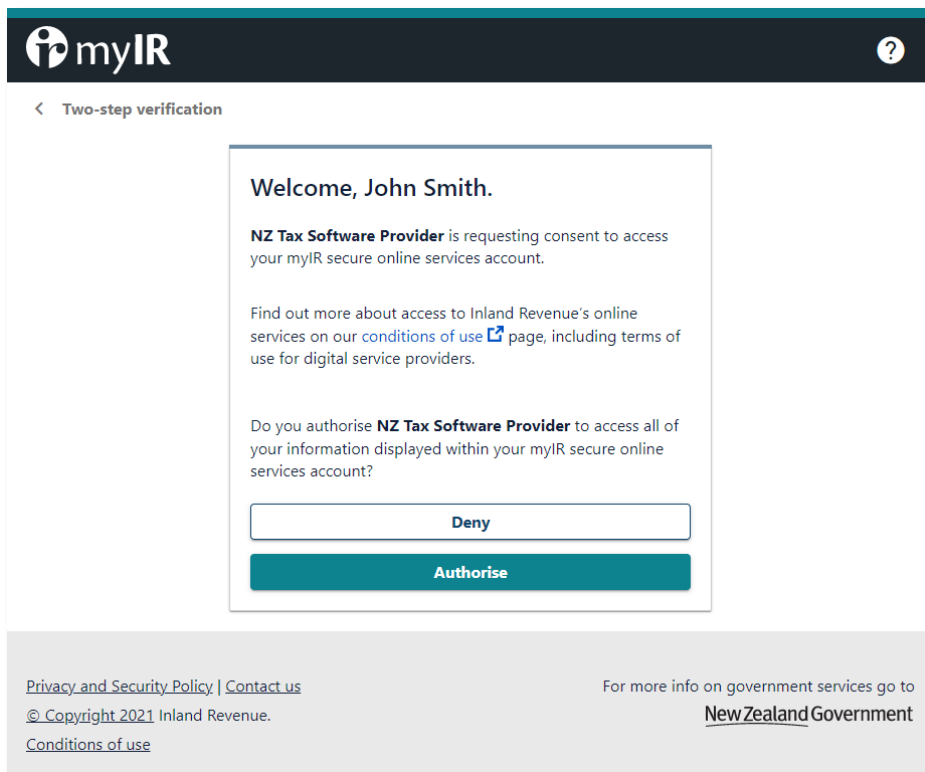
```
{
  "error": "invalid_request",
  "error_description": "Invalid request format. Missing parameter: response_type"
}
```

#### 2.1.3.4 Consent Page (step 4)

The following consent screen will be presented to the user the very first time they authenticate with their IR credentials via Step 3 above. This screen will not be presented for subsequent logons, unless 3<sup>rd</sup> party consent it revoked in myIR.

The user must click Authorise for the process to proceed. If they click deny then the process stops, and the authorisation server will not respond with an authorisation code as described in Step 5 below.

```
https://client.example.com/OAuthWebhook?error=access_denied&state=xyz
```



**myIR** ?

< Two-step verification

Welcome, John Smith.

**NZ Tax Software Provider** is requesting consent to access your myIR secure online services account.

Find out more about access to Inland Revenue's online services on our [conditions of use](#) page, including terms of use for digital service providers.

Do you authorise **NZ Tax Software Provider** to access all of your information displayed within your myIR secure online services account?

Deny

Authorise

[Privacy and Security Policy](#) | [Contact us](#)  
 © Copyright 2021 Inland Revenue.  
[Conditions of use](#)

For more info on government services go to [New Zealand Government](#)

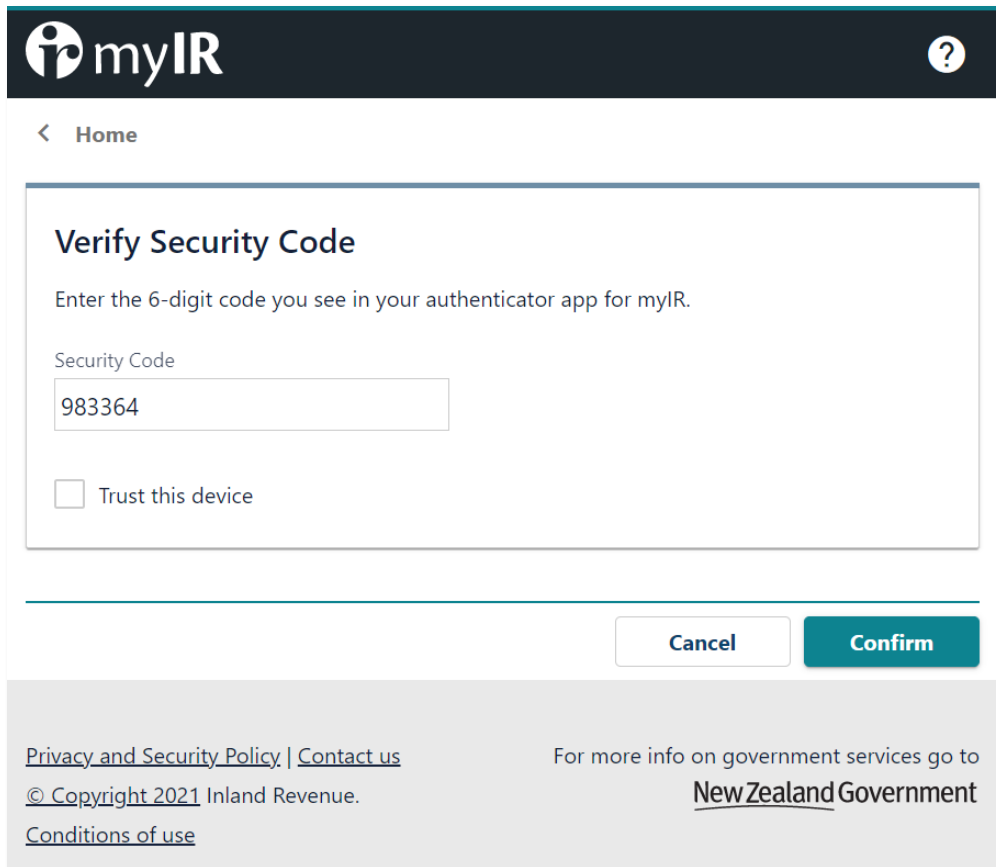
When testing, it is recommended to revoke consent in order to ensure that the Service Provider

software is correctly managing the Consent Management page being presented and Authorisation given.

Please contact [GatewayServices@ird.govt.nz](mailto:GatewayServices@ird.govt.nz) for revoke consent assistance.

**Note:** If the user has opted to enable Two-Step verification on their myIR User ID, they will be presented with an extra step to enter a 6-digit security code.  
No additional coding expectation by the Digital Service Provider are required to handle Two-Step verification.

Example of the Verify Security Code page to be used in combination with Authenticator Apps



The screenshot shows the myIR user interface. At the top is a dark header with the myIR logo and a help icon. Below the header is a navigation bar with a back arrow and the word 'Home'. The main content area is titled 'Verify Security Code' and contains the instruction: 'Enter the 6-digit code you see in your authenticator app for myIR.' There is a text input field labeled 'Security Code' containing the value '983364'. Below the input field is a checkbox labeled 'Trust this device'. At the bottom right of the form are two buttons: 'Cancel' and 'Confirm'. The footer contains links for 'Privacy and Security Policy', 'Contact us', and 'Conditions of use', along with copyright information for 2021 Inland Revenue. On the right side of the footer, it says 'For more info on government services go to New Zealand Government'.

myIR

< Home

### Verify Security Code

Enter the 6-digit code you see in your authenticator app for myIR.

Security Code

☐ Trust this device

Cancel Confirm

[Privacy and Security Policy](#) | [Contact us](#)  
© Copyright 2021 Inland Revenue.  
[Conditions of use](#)

For more info on government services go to  
**New Zealand Government**

### 2.1.3.5 Respond with Authorisation Code (Step 5)

If successful, the authorisation service will respond with the Authorisation Code to the Digital Service Provider `redirect_uri` as described below:

```
https://client.example.com/OAuthWebhook?code=eyJhbG...rWWk8hbs_o6uY&state=xyz
```

Name	Description	Valid Values
<b>Code</b>	Authorisation code. The authorisation code is a code that can be presented to the token endpoint in exchange for access and refresh tokens. Authorisations codes cannot be used more than once. If a digital service provider attempts to reuse an authorization code, the provider will be returned a Bad Response error.	Encrypted string ~100 characters
<b>State</b>	Digital Service Provider's defined state. The use of the state parameter is strongly recommended for additional security. The value should relate to the logged-in user and allows the Client (software provider) to confirm that the request being received at their Redirect URI is valid. See <a href="https://tools.ietf.org/html/rfc6749">https://tools.ietf.org/html/rfc6749</a> for more information (Section 12.12).	Must be < 200 characters.  Allowed Characters: a-z A-Z 0-9  - . ? , : / \ + = \$ #  Not space (%20, (ASCII 32))

If not successful, a response is sent with an HTTP code and a JSON payload containing the error code and error description.

Errors are:

HTTP Status Code	Error Code	Error Description
<b>400</b>	<b>invalid_request</b>	Invalid request format. Missing parameter: <code>redirect_uri</code>
	<b>invalid_request</b>	Invalid <code>redirect_uri</code> . Provided <code>redirect_uri</code> [...] is not configured for this client
	<b>invalid_request</b>	Invalid request format. Missing parameter: <code>response_type</code>
	<b>invalid_request</b>	Invalid <code>response_type</code> . Response type must be 'code'
	<b>invalid_request</b>	Invalid request format. Missing parameter: <code>client_id</code>
	<b>invalid_client</b>	Client is invalid.
	<b>invalid_request</b>	Invalid request format. Missing parameter: <code>scope</code>
	<b>invalid_scope</b>	Invalid scope requested

### 2.1.3.6 Request Access Token (Step 6)

Once an Authorisation Code has been returned to the client application it must be exchanged for an OAuth Access Token by doing an HTTPS Post to the Token Service as follows:

(Example is for one IR Test environment – see 2.1.9 below for all endpoints)

```
https://test5.services.ird.govt.nz/gateway3/oauth/token
Authorization: NTQzMjFpZ...ZWxjb21lMQ==
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

<form>
  grant_type=authorization_code
  redirect_uri=http://localhost/webservices/Oauth/SignIn
  code=kf77pbtw3j55kxbgf3t3ygd9379htr88mxxnm2yrhth7v39hxr63g8yx3xkzj2f6kyjx364dsk58z
  899sht94pb7q5767txfn3h
</form>
```

Name	With the values of: Description	Required	Valid Values
<b>redirect_uri</b>	The Client application's redirect URI for the Authorisation Token.	Required	Business Partner defined & agreed with IR
<b>grant_type</b>	The grant type is authorization_code	Required	authorization_code
<b>code</b>	Authorisation Code as supplied by the authorisation service in step 4	Required	Encrypted string ~100 characters
<b>PKCE Support – See section <a href="#">2.1.8</a> about PKCE</b>			
<b>code_verifier</b>	The code verifier for the PKCE request, that the app originally generated before the authorization request.	Optional	This is a cryptographically random string using the characters A-Z, a-z, 0-9, and the punctuation characters -. _ ~ (hyphen, period, underscore, and tilde), between 43 and 128 characters long

The header fields must contain the Client ID and Client secret and content type:

```
Authorization: Basic NTQzMjFpZ...ZWxjb21lMQ==
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

With the values of:

Name	Description	Required	Valid Values
<b>Authorization</b>	"Basic" + Base64 encoded (ClientID + ":" + Client Secret)	Required	Base64 encoded string

<b>Content-Type</b>	Content type	Required	application/x-www-form-urlencoded; charset=UTF-8
<b>Content-Length</b>	Content length of the POST request	Required	Number of bytes of data in the body of the request

The response contains the OAuth Access Token – this must be passed on subsequent service calls.

If the client is registered for Refresh Tokens, this will also be returned at the point, see Section b) Refresh Token Service.

The error response should not be used within client code to determine the course of action. The error messages are informational, intended to be logged for human review during integrating with IRD.

The structure of the error response body is:

```
{
  "error": "Type of error",
  "error_description": "Description of the error"
}
```

HTTP Status Code	Error Code	Error Description
<b>400</b>	invalid_request	Invalid client. Missing authorization header.
	invalid_request	Invalid request format. Missing parameter: grant_type
	invalid_request	Invalid authorization header.
	invalid_client	The provided secret or assertion are not valid for this client.
	unsupported_grant_type	Invalid grant_type.
	invalid_request	Invalid request format. Missing parameter: code
	invalid_request	Invalid request format. Missing parameter: redirect_uri
	invalid_request	Invalid authorization header.
	invalid_client	Client is invalid.
	invalid_request	This API requires authentication using HTTP Basic Auth or by including credentials in the request body.
	access_denied	The provided secret or assertion are not valid for this client.
	invalid_grant	Refresh token is invalid.
<b>401</b>	invalid_grant	Invalid authorization code.



	invalid_grant	The authorization code has expired.
	invalid_grant	Invalid redirect_uri. Value does not match the authorization request.

#### 2.1.4 Refresh Token Service

In normal use it is expected that the Access Token will be re-used while it is active, and the client will make several calls using this Access Token. If the Access Token has expired, the client can request another Access Token using the Refresh Token and client credentials.

It's critical for the most recently issued refresh token to get immediately invalidated when a previously used refresh token is sent to the token service. This prevents any refresh tokens in the same token set from being used to get new access tokens. This is known as a "Replay Attack".

Automatic reuse detection is a key component of a refresh token rotation strategy. START IAM has already invalidated the refresh token that has already been used. However, since START IAM service has no way of knowing if the legitimate software is holding the most current refresh token, it invalidates the whole token set to ensure security is maintained.

The Refresh Token is granted as part of the Cloud Authorisation Service, the client will need to hold this and when required can be exchanged for a new Access and Refresh Token granting longer term use.

**Note:** Native Desktop Application will not be issued a Refresh Token. The user will be required to re-authenticate to obtain a new Access Token.

The API call to exchange a Refresh Token for an Access token takes the form:

```
https://test5.services.ird.govt.nz/gateway3/oauth/token

<form>

    grant_type=refresh_token

    &refresh_token=<refresh-token-value>

</form>
```

With the values of:

Name	Description	Required	Valid Values
<b>grant_type</b>	The grant type is required and takes the value refresh_token	Required	refresh_token
<b>refresh_token</b>	The refresh_token field contains the Refresh Token	Required	<Refresh Token>

The header fields must contain the Client ID and Client secret and content type:

```
Authorization: Basic          NTQzMjFpZ...ZWxjb211MQ==
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

With the values of:

Name	Description	Required	Valid Values
<b>Authorization</b>	"Basic " + Base64 encoded (ClientID + ":" + Client Secret)	Required	Base64 encoded string
<b>Content-Type</b>	Content type	Required	application/x-www-form-urlencoded; charset=UTF-8
<b>Content-Length</b>	Content length of the POST request	Required	Number of bytes of data in the body of the request

The normal expected response if the Access Token is valid is a new Access and Refresh Token.

**Note:** Refresh tokens are valid for only 1 year. Refresh tokens can only be used once when a new Access Token has been requested and the Customer consent is still valid.

Refresh Token Errors:

HTTP Status Code	Error Code	Error Description
<b>400</b>	invalid_request	Invalid authorization header.
	invalid_client	Client is invalid.
	invalid_request	This API requires authentication using HTTP Basic Auth or by including credentials in the request body.
	access_denied	The provided secret or assertion are not valid for this client.
<b>401</b>	invalid_grant	Refresh token is invalid.

### 2.1.5 Introspection Token Service (Validate)

This service will validate an Access or Refresh Token. The following example is for one IR Test environment – see 2.1.9 below for all endpoints.

```
https://test5.services.ird.govt.nz/gateway3/oauth/introspect
<form>
```

```
token=<token-value>|<refresh-value>

&token_type_hint=access_token | refresh_token

</form>
```

With the values of:

Name	Description	Required	Valid Values
<b>token</b>	The token field contains the Access Token or Refresh Token	Required	<Access Token>   <Refresh Token>
<b>token_type_hint</b>	A hint about the type of the token submitted for introspection.	Optional	access_token   refresh_token

The header fields must contain the Client ID and Client secret and content type:

```
Authorization: Basic NTQzMjFpZ...ZWxjb21lMQ==

Content-Type: application/x-www-form-urlencoded;charset=UTF-8
```

With the values of:

Name	Description	Required	Valid Values
<b>Authorization</b>	"Basic " + Base64 encoded (ClientID + ":" + Client Secret)	Required	Base64 encoded string
<b>Content-Type</b>	Content type	Required	application/x-www-form-urlencoded;charset=UTF-8
<b>Content-Length</b>	Content length of the POST request	Required	Number of bytes of data in the body of the request

Successful responses will return HTTP Status code 200 with JSON response.

```
{
  "active": true,
  "client_id": "clientID",
  "username": "myIRUsername",
  "scope": "MYIR.Services",
  "sub": "545378fc-60fe-4a88-b638-12a5950a2201",
  "exp": 1658144943,
  "iat": 1658116143
}
```

Introspection Token Service Errors:

HTTP code	Error Type	Description
<b>400</b>	invalid_request	Invalid request format. Missing parameter: token
	unauthorized_client	Token refresh is not allowed for this client.
<b>401</b>	invalid_client	Your client must authenticate to use this API.
	invalid_client	Invalid authorization header.

### 2.1.6 Revocation Token Service (Revoke)

A token revoke process is available that will allow the client to revoke the acquired access token or refresh token.

Customers may also revoke their consent through a function in IR Online Services, "myIR". The following example is for one IR Test environment – see 2.1.9 below for all endpoints.

```
https://test5.services.ird.govt.nz/gateway3/oauth/revoke
```

```
<form>

    token=<access_token> | <refresh_token>

    &token_type_hint=access_token | refresh_token

</form>
```

With the values of:

Name	Description	Required	Valid Values
<b>token</b>	The token field contains the Access Token or Refresh Token	Required	<Access Token>   <Refresh Token>
<b>token_type_hint</b>	A hint about the type of the token submitted for introspection.	Optional	access_token   refresh_token

The header fields must contain the Client ID and Client secret and content type:

```
Authorization: Basic NTQzMjFpZ...ZWxjb21lMQ==

Content-Type: application/x-www-form-urlencoded;charset=UTF-8
```

With the values of:

Name	Description	Required	Valid Values
<b>Authorization</b>	"Basic" + Base64 encoded (ClientID + ":" + Client Secret)	Required	Base64 encoded string
<b>Content-Type</b>	Content type	Required	application/x-www-form-urlencoded

<b>Content-Length</b>	Content length of the POST request	Required	Number of bytes of data in the body of the request
-----------------------	------------------------------------	----------	--

Successful responses will return HTTP Status code 200 with no content.

Revocation Token Service Errors:

HTTP code	Error Type	Description
<b>400</b>	invalid_request	Invalid request format. Missing parameter: token
<b>401</b>	invalid_client	Invalid request format. Missing parameter: client_id
	invalid_client	Invalid authorization header.

### 2.1.7 Security Considerations

Protecting the integrity of the Client Secret is an important requirement for providers, the exact implementation is left to the provider, but it must not be stored in plain text either in the web, mobile, or desktop application. Our preference is for this to be stored on a back-end server and made available to the Digital Service Provider's application.

If the Client Secret is compromised, it shall be invalidated and a new secret issued.

Token Type	Lifespan	String Lengths	Rationale
<b>Authorization Code</b>	10 minutes	100 characters	RFC encourages short auth code expiry and industry leaders use 10 minutes as a default.
<b>Access Token</b>	8 hours	JWT are variable length	Reflects a standard business day.
<b>Refresh Token</b>	1 year	50 characters	Reflects the standard use case of an individual filing income tax every year.
<b>Consent</b>	5 years	n/a	Business decision for all resource owners to reauthorize after 5 years.

**Note:** Each new logon by a user will result in a new Access and Refresh token set. Service Providers need to ensure adequate care is taken when storing these tokens.

### 2.1.8 PKCE Support

Proof Key for Code Exchange (or PKCE) will be supported on the authorization service. The authorization service will store the code\_challenge sent by the client application in the authorization code request. Before issuing the Access token, the authorization service will compare the code\_verifier sent by the client application in the access token request to the code\_challenge to ensure that the authorize code has not been compromised. This feature is optional and cannot be enforced per digital service providers.

More information can be found on the oauth.net website [PKCE for OAuth 2.0](#)

### 2.1.9 Endpoints

Endpoints for the token-based Authentication and Authorisation Service are as follows:

**Test Environment:**

Authorisation Code: <https://test5.services.ird.govt.nz/gateway3/oauth/authorize>  
Token: <https://test5.services.ird.govt.nz/gateway3/oauth/token>  
Introspection: <https://test5.services.ird.govt.nz/gateway3/oauth/introspect>  
Revocation: <https://test5.services.ird.govt.nz/gateway3/oauth/revoke>

**Production Environment:**

Authorisation Code: <https://services.ird.govt.nz/gateway3/oauth/authorize>  
Token: <https://services.ird.govt.nz/gateway3/oauth/token>  
Introspection: <https://services.ird.govt.nz/gateway3/oauth/introspect>  
Revocation: <https://services.ird.govt.nz/gateway3/oauth/revoke>

**Note:** The test5.services.ird.govt.nz environment does not require IP address whitelisting. As an additional security measure IP address whitelisting is required for Pre-prod environments containing production data. New splash page requesting user credentials will be presented in case of no IP address whitelisting. Please contact Gateway Services for further information.

---

## 2.2 Native Application Token Auth

The OAuth 2.0 transaction flow described above will not change for Service Providers running Native Applications. A Native Application (commonly called a Native App) is an application that is installed by the user to their device or a desktop application, as distinct from a web app that runs in the browser context only.

IR has a duty to protect customer information in transactions. In order to meet this obligation, the security and traffic controls for Native App endpoints are more stringent. This is due to the more public nature of these endpoints and the inability to identify the caller by an X.509 certificate as per the cloud connection.

To further this end, IR is adopting [RFC8252 OAuth 2.0 for Native Apps](#)

Your attention is drawn to the best practice description in section 1 of using an external browser for OAuth by Native Apps.

IR will be providing separate endpoints when using gateway services with applications using Native App OAuth applications. Mutual TLS will not be used for these endpoints so no X.509 client cert will be required but server-side TLS will still be used.

Please refer to the Build Pack for each respective service for further details.

Note the endpoints for the Native App OAuth calls will not be changing from those outlined in Section [2.1.9](#) above.

A Client ID and secret will still be issued, the Client ID will need to show both the software vendor and application name e.g., SmartSoftware\_SmartPay or SmartSoftware\_SmartSoftware.

At registration IR will note the OAuth user ID is type 'Native App' and not 'Normal'. The refresh token capability will not be offered for Native App OAuth tokens.

Special redirect will be allowed and the three redirection schemes in section 7 of RFC 8252 will be supported. IR does not limit or put a preference on any of these.

If considering 7.3 then attention is drawn to recommendations available on the internet regarding the use of 127.0.0.1 rather than localhost. See [ietf.org](http://ietf.org).

7.3 of the RFC 8252 also states:

*The authorization server MUST allow any port to be specified at the time of the request for loopback IP redirect URIs, to accommodate clients that obtain an available ephemeral port from the operating system at the time of the request.*

This is not currently supported by the IR. As a work-around IR will be able to provide a selection of five or so port numbers to be agreed with the Service Provider and registered in IR's back-end OAuth system or via the Developer Portal. Please discuss with your IR Account Manager

---

## 2.3 SSH Keys

This authentication and authorisation mechanism is used in SFTP file transfers to identify the respective organisations sending/receiving files, in this case IR and the Service Provider.

SSH Key authentication and authorisation will be used for file transfers in which SFTP 3.0 is used. This version of SFTP requires the use of SSH version 2.0.

The public key algorithm for SSH authentication keys must be ECDSA with a minimum field/key size of at least 160 bits.

Certain FTP file transfers will also require payload encryption and signing to ensure that once a file is transferred to an endpoint only an authorised party can interpret it. This is optional and the need for this will be identified in the respective On-boarding pack for a file transfer.

The obligation to encrypt the transferred files is defined in the NZISM under the classification privacy ratings. These are based upon the sensitivity of the data, the harm that might be caused if the data is disclosed, along with considerations such as the volumes being transferred. An assessment of these factors is required when implementing new file transfers and when adding content or changing the parties to any existing file transfers.

For files from IRD to partners PGP encryption will use Advanced Encryption Standard (AES) with a 256-bit key and the PGP hashing will use Secure Hash Algorithm (SHA) SHA-256.

Note the IR reserves the right to upgrade cryptographic standards (e.g. algorithms and key lengths) in response to factors such as external threats, industry standards and changes to NZISM. IR's partners will be consulted, via IR's relationship managers, about changes of this nature.

Currently IR has the ability to push and pull files being exchanged, but the Service Provider always hosts the FTP server.

## 2.4 M2M using Client Signed JWT

Inland Revenue's machine-to-machine (M2M) authentication mechanism will use Client Signed JSON Web Tokens (JWT).

When applying this pattern, the external parties fulfil the following roles:

- **Resource Owner** – this is the party under whose identity the transaction is being undertaken. This has the same meaning as the Resource Owner in the OAuth protocol. The Resource Owner's identity binds to a myIR user ID within START, and from this to START's authorisation rules and the user's access rights. Each service/API call is verified and trusted because it is digitally signed with a public/private key pair belonging to the resource owner using an approved signing algorithm (currently RSA or preferably ECDSA). The resource owner's public key is exchanged during the on-boarding process.
- **Service Provider** – this is the party who operates the application that consumes IR's gateway services. This is the equivalent of the Client Application when using the OAuth protocol. The service provider encrypts the data that they exchange with IR.

There are two permutations:

- a) A Service Provider can interact with IR on behalf of one or more Resource Owners, this is a typical software-as-a-service or outsourced operating model. Possible examples include:



- A payroll intermediary is a service provider to many employers (resource owners).
- An accounting platform (a service provider) can have many clients including tax agents and private clients (resource owners).
- A KiwiSaver Administrator (a service provider) typically operates on behalf of many KiwiSaver Scheme Providers (resource owners).

b) An organisation can be both the Resource Owner and Service Provider, for example:

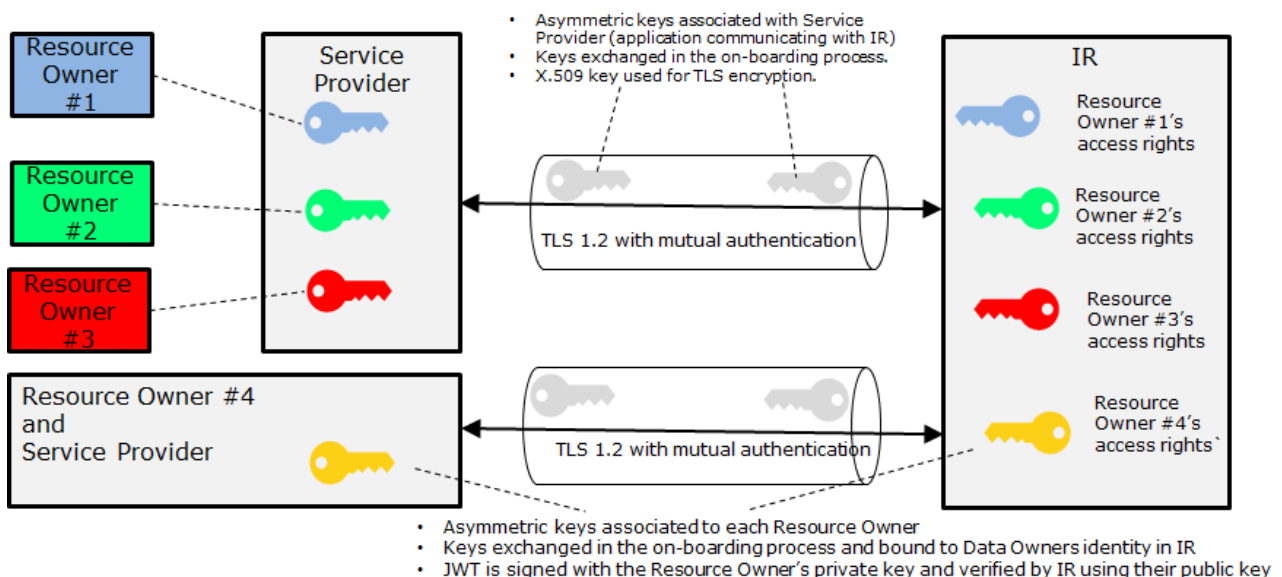
- A large corporate can operate their own payroll system.
- An accounting firm can operate their own accounting software.
- Some large banks operate their own KiwiSaver schemes and platform.
- Most government agencies fulfil both roles.

Whenever implementing this M2M authentication mechanism it is important that these roles are clearly identified.

The JWT will be created by the party calling Inland Revenue and, along with other standard JWT information detailed below, will contain the startLogon parameter to identify the myIR logon. The header and payload will be base64 encoded and then signed with the private key of the Resource owner.

The Resource owner will create a signing certificate and from that a private/public key pair. (See Appendix C). They will share the public key with IR as part of the on-boarding process. The Resource Owner will use the private key for signing the JWT and IR the public key for verifying the signature as described below.

This topology is depicted in the following diagram:



As a result, IR will know who the token comes from (the Resource Owner), and from interrogating the token will also know the myIR logon from the startLogon value. IR will use this myIR logon information to apply the delegations' model already existing in eServices.

Where startLogon is null, then the Resource Owner's relationships inherent in START will be used to determine the information available to the calling service. This is done by mapping the Client Signed certificate thumbprint to the Customer in START.

The hierarchy of use is the startLogon and then Customer. If the start\_Logon is populated, then the delegations of the myIR logon will be used. If the start\_Logon field is not populated, then the Customer delegations in START will be used.

The services using this token will be sent on the mutual TLS (mTLS) 4046 service endpoint.

Using mutual TLS will enable IR to use the Software Provider's mTLS X.509 public key to recognise the Service Provider but will rely on the JWT token to identify the resource owner.

Gateway Services will use this token in the HTTP header of a message in the similar manner that an OAuth token has been used, namely:

`"Authorization: {JWTAccessToken}"`

Note that the word 'Bearer' (used with the OAuth token) is not present.

JWT consists of three parts separated by dots. i.e. header.payload.signature.

The header and the payload JSON are each separately Base64Url encoded.

The token to be used for interacting with Inland Revenue will be as below:

```
Base64Url encoded {  
  "alg": <algorithm value>,  
  
  "typ": "JWT",  
  
  "kid": "M2M"  
}  
.  
Base64Url encoded {  
  "sub": <signing certificate thumbprint value>,  
  
  "iss": <issuer value>,  
  
  "startLogon": <myIR_user>,    (can be null, see below)  
  
  "iat": <epoch issued value>,  
  
  "exp": <epoch expired value>  
}  
.  
JWS Signature (  
  base64UrlEncode(header) + "." + base64UrlEncode(payload)  
)
```

With the values of:

Name	Description	Required	Valid Values
<b>sub</b>	Subject (whom the token refers to).	Required	Value will be the thumbprint/fingerprint of the JWT signing certificate sent to Inland Revenue. e.g. d3e80pl7ba8858d994094c05a208e9684e6f08ba
<b>alg</b>	Signature or encryption algorithm.	Required	RS256, RS384, RS512 ES256, ES384, ES512
<b>typ</b>	Type of token.	Required	JWT
<b>kid</b>	Key ID.	Required	M2M
<b>iss</b>	Issuer who created this token.	Required	Value agreed between issuer and IR e.g. www.name.com
<b>startLogon</b>	The myIR logon of a representative of the token subject. The Subject must be the data owner.	Optional	Valid myIR log-on (the user, NOT the password). The GWS transaction using this token will inherit the delegations this myIR user has in eServices.  This can be null in certain circumstances when the calling Data Owner's relationships determines the delegation rights for the calling service. Refer to individual Service Build packs to understand when this applies.
<b>iat</b>	Issued at (seconds since Unix epoch).	Required	Must not precede the signing certificate issue date e.g. 1560144847
<b>exp</b>	Expiration time (seconds since Unix epoch).	Required	Must not exceed 8 hours from the iat issue time value

Refer to [RFC 7518](#) for a specification regarding cryptographic algorithms and identifiers to be used with the JSON Web signature (JWS).

The value stated in the "alg" parameter of the JWT header determines the algorithm used to generate the digital signature.

"alg" Param value	Digital Signature	Implementation Requirements
<b>RS256</b>	RSASSA-PKCS1-v1_5 using SHA-256	Recommended
<b>RS384</b>	RSASSA-PKCS1-v1_5 using SHA-384	Optional
<b>RS512</b>	RSASSA-PKCS1-v1_5 using SHA-512	Optional
<b>ES256</b>	ECDSA using P-256 and SHA-256	Recommended+

<b>ES384</b>	ECDSA using P-384 and SHA-384	Optional
<b>ES512</b>	ECDSA using P-521 and SHA-512	Optional

The use of "+" in the Implementation Requirements column indicates that the requirement strength is likely to be increased in a future version of the specification.

The following points must be observed by the token creator:

- Key generation must be on a secure, trusted host, and the issuer must always protect their private key.
- Different keys are required for test and production.
- Character set UTF-8.
- Certificate needs to meet the X.509 V3 standard.
- Signing Algorithm: ECDSA is preferred but RSA will be supported.
- Validity period: up to 4 years.
- Information in Subject attribute should match the organisation.
- Certificate format: \*.crt, \*.cer or \*.pem .
- Self-signed certificates are permitted for signing certificates provided that the source of the certificate is confirmed to be a valid representative of the resource owner.
- Refer to Appendix C for example code and a completed token.

Note that Inland Revenue reserves the right to upgrade cryptographic standards (e.g. algorithms, key lengths, token expiry, protocol versions etc) in response to factors such as external threats, industry standards and legislative change. IR's partners will be consulted, via IR's relationship managers, about changes of this nature.

## 3 TLS & Mutual TLS Authentication

### 3.1 Ciphers

Inland Revenue supports the following TLS ciphers listed below:

TLS	IANA Cipher Name	OpenSSL Cipher Name
<b>TLS1.3</b>	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384
<b>TLS1.3</b>	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256
<b>TLS1.3</b>	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256
<b>TLS1.2</b>	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDH-ECDSA-AES256-GCM-SHA384
<b>TLS1.2</b>	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDH-ECDSA-AES128-GCM-SHA256
<b>TLS1.2</b>	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	ECDHE-ECDSA-CHACHA20-POLY1305

### 3.2 Mutual TLS Authentication (mTLS)

We operate in a typical CA trust-based authentication model (utilising mTLS) so that any certificate that contains a pre-approved Common Name (CN) and is issued only by one of IR's approved trusted root Certificate Authorities will be accepted.

You need to ensure a Common Name is generated in the Developer Portal and used as part of the process of ordering a certificate from the approved list of CA's below:

- Amazon
- Comodo
- Digicert
- Entrust
- GeoTrust
- Let's Encrypt
- Sectigo
- Thawte

DSP Common Name will be assigned through the IR Developer Portal, and will use the DSP's registered DNS domain, prefixed with a unique security identifier assigned by IR, for example:

- 298f9c17bbbe48958994982c383c409c.irdgws.yourdomainname.com
- 298f9c17bbbe48958994982c383c409c.irdgws.test.yourdomainname.com
- 298f9c17bbbe48958994982c383c409c.irdgws.prod.yourdomainname.com

NOTE: The assigned Common Name has to match exactly with your certificate and what is loaded on our systems.

NOTE: Separate certificates are required to access test and production environments. We request you only use this Common Name certificate for mTLS with IR and this certificate is not used for other messages exchanges or web sites you may have.

## 4 Appendix A – Sample payloads

## 4.1 Request Authorisation Code

Service consumer to IR. Refer to step 2 in section 2.1.1.

### 4.1.1 Request

Service provider will send a HTTP GET request to Auth Server, url format as below:

[https://test5.services.ird.govt.nz/gateway3/oauth/authorize?client\\_id=Test99999999&redirect\\_uri=https://myreturnuri/test/&scope=MYIR.Services&response\\_type=code&state=123](https://test5.services.ird.govt.nz/gateway3/oauth/authorize?client_id=Test99999999&redirect_uri=https://myreturnuri/test/&scope=MYIR.Services&response_type=code&state=123)

## 4.2 Authorisation Code response

IR to Service consumer. Refer to step 4 in section 2.1.1.

#### 4.2.1 Success Response – Authorisation Code sent

https://myreturnuri/test/?code=b2pZWt3RUZ4eXU3UnJmZjhHYkRXT09fnl3SkJuR2FnWDhTdVlSNEkvR0hhejcyUU55Vtc2bGJmODgxcmlROUXY3UXV0RG0zaURNTUc2T2NjOHRrb3I5Mm1YSE1kMHftN202VFRFejlLUXJaa2dGMFJSNUZDTDZzNVZKeHhORGxoWEd5cGhCV2lYV1l1UDVxUVFak2I4STaXTeH2WWE0MVU1bmE0TFZORUdlQzJuVC9qVys1T2J5RjEvWmQ5OVpmY3hWU3B3ZUcrdk5Ka2ttbFhVNExmDg1wbE5KYlZMd2dTU1k4SVNFVDV4MStJdm04d2tQOWlhQVE2b2ZRY2lSZVAzQW0xNkFsbkh6eHdNNitCaFpsWU55ZzJna3ZSb095WkRSeERKMWF1NTRzV3lSTGJhWU1Sb1FOYTdRuk1JNXdUZWPfNnhsVmc1ZEd2QUxrWld0Qmhpct1vtKVRWlB2anRubUJ39XUFhpRnFIQ0dwWjZDbmRaY0FLQWFJYy9ScFRsQ0pLSFFRRjduZjVLU3ZtVXc4TVFsbWsrNcl1I5V2d6RGXpNW5DandpR2lrK0l0UEY4S3RPMjFRRTZLNnAydMbnVnW54S0ZQOS9WZFJEMG90CxlLdc9Oc1h4T1FLYUR0Y3FXOXFpemlHcXNUVUw2ZGM5cw1DMWDHNl1YK0pZwK5EbFhRdVi0RFVvWmRtd0tZcVkhVOE9VRVUYaG13a2FQblkybn1Zb2pBbkG2QkdQYUpIdmtrTxxEcVdRbVp0Zm8yaEM2UkZhVzBDUW1uN1pUMVZmS3grcXhmMDVST29kRW16Y3c0SEdheUFRYWVlUUE9PD==&state=123

### 4.3 Request Access token

Service consumer to IR. Refer to step 5 in section 2.1.1.

#### 4.3.1 Exchange Authorisation Code for OAuth Access Token.

At this step as a service provider, you need to send a HTTP POST request to Auth Server with Authorization Code and your client credentials.

URL format:

<https://test5.services.ird.govt.nz/gateway3/oauth/token>

Request headers:

```
Authorization: Basic VGVzdMwMjA2NDkyOk9hdXRoMk1SU2VjcmV0dA==]
Content-Type: application/x-www-form-urlencoded
Content-Length: 1090
```

Request body:

[illegible]

```
Tc1LTQ3ZmYtODk0NC1hNTcxZWNLyZgzNmYiLCJvcMfjbGUub2F1dGguc2NvcGUiOiJNWU1SL1N1cnZpY
2VzIiwib3JhY2xlM9hdXR0LmNsaWVudF9vcmlnaW5faWQiOiJUZXR0MzAyMDY0OTIiLCJlc2VyLnRlb
mFudC5uYW1lIjoiRGVmYXVsdERvbWVpbiIsIm9yYWNsZS5vYXV0aC5pZD9kX2lkIjoiMTIzNDU2NzgtM
TIzNC0xMjM0LTEyMzQtMTIzNDU2NzgtMDEyIn0.NTBu3R-JwaaOWfvMdwAHqY7Ji3YI3I-
bSTXq6jauqEUhswLmAG6cbpGaSky50ECbHNv2skU8WVZ0RYv67KPgITGXJz0ZKSjqOgiZ0R4kFCZ7as
N8yjlzXgxwWk4mPXL5E02u24-VMbr_hrNZYDZbakOpz4uY6U1SSNECmw0ac8
```

#### 4.3.2 Success Response – Access Token sent

The response contains the Access Token and expiry time. If refresh tokens are used this is also returned:

```
{
  "access_token":
    "eynusrmrijixijcszbiiuogh5ccci6Fmi0Xcztpscpmiuvil6Ijm5Qem5Qjzyzebbmdn4Q0Dwvtquaqrordg5Evqt
    zqzuoirzmulIqqfur.mjiyuoyjeey2Tny0Ujn0Mzspmi0isa6Tiojgzy4Kzm3Lhwnf2Uyymstnd04Njgtqwl5M2
    E5Ymzxm2Mcmwcqizisiyl6Mih0dhvoz1wxbc90Nzx0Sm5xjzz2nwyac15quzpmc292dc5vuelwz2F0Hztxeldnc29
    xxcrdoh8iilcxjqepjoy2Ytn0Nju3Snizimn1Iiy6Ije3Kygwjntrrytnnmlkmyt0Smldyvtmhkkljemhtmw1Icon
    symmnin0Xjy0Gt9bn24ujioib21Ub20cmimjxlyzj29wzsi6Ki1Sizuuv2Gbdplllzmnxjyjiccijmrlvb6Ihcmz
    3Icqlhrwlohdhpjdwzjoicclgdezvcdduwv2Lvnvcujxznbymdpujdqmmmlalkdx2dvhdg3Cykxxz9yvxv0wivwia
    fiz3VbviojnbthjriqilSfwjkuuj9Xo0Ienpjjaehoclulie2Ytn00qt1Jzm9.a-
    px0Kmcfvbolessrgdoycqyk7Uylcj_Qds7Vi0qnkjfxbdlujxn30rlneuuvpslambz20qxoc_Ctvjshpevrnk6_91
    38ixyvkdqzvg_xk4mvaxt-Scqlgl-pkhbhqdnbdpiwv30g78-
    Vvqdgza3Aya2Tulak4dwebaq2Ckcgqyt70tckfs6kvhvcgn5Fotfohbtueqnxxtagopmvtfwrxyfl1Xwthaspjqoj
    1S6_cmt4I0vrcm3Jmc4haiv-Vzrmndqeh6x5Xhvlmh_x7oyldgthms4Jl7zenq3feyj25Sh-
    Xlw6swf4Sdw4g5tb0gxpobpaas005zpcqphc9Yw",
  "token_type": "Bearer",
  "expires_in": "28800",
  "scope": "MYIR.Services",
  "refresh_token": "n5zc5b8h|ty6kvqbx7yqrc6fqw8knczt435nm49th97d6mxgqj2"
}
```

#### 4.4 Request Refresh token

Service consumer to IR. Refer to section 2.1.4.

##### 4.4.1 Refresh request

At this step as a service provider, you need to send a HTTP POST request to the OAuth Server with the Refresh Token and your client credentials.

Request headers:

```
Authorization: Basic VGVzdDMwMjA2NDkyOk9hdXR0Mk1SU2VjcmV0dA==
Content-Type: application/x-www-form-urlencoded
Content-Length: 1127
```

Request body:

```
grant_type=refresh_token
&refresh_token=6fgyc4qz4nkk85q9rcmqh67v8jzmxxy7b9tt6xb553nqcdwhtn2
```

##### 4.4.2 Refresh token reply

The Refresh token reply is the same as an authorisation code exchange for an Access token.

```
{
  "access_token": "lxhaM5QQFYNEYYYgQJSND9FVphT2GkERXQ2NkrSCLbbbZLYZMZwUEENUcJ9LarOW
  y3yZYTcEVQYV446E2Uf6YEMAMQRRWvyCS4KUHIIIIIICRE5LgOCvC2BZwI2GC82TTbeznaE2jzBNQzD
  fHMDgXXXVZcZVOLLLLSW8VCaMRR6hMMNLfNND0Jd8TjjjDNOu4MVF2nYjDm264cwagTIVddNbrbWdde
  iCSCSCyXANAvO66w6X6666ssssssSCCiVTXXk39nli3iij3CCXiCyCWhvpqKJcccXiiiimT5jk0wk5ku
  pPWjWhB03500xma2xXjjxhTXm5zXnnhP5t2ytVcN1220Tkx000kWW13UzLfyhyVwz9pSiiiiizo7dtK
```

```
zFrR9ff4U-EpGzcJ3nWXU5Jmn2051LlnKUD1Ec1SSAWTjmJTXnHPMMeZaMNJZJJZfZ-
ENQY0RRYeLLNNMdAjudS-
QROOR1JCrMLXzSXGBMX6UDSR0Wu22GeAOHBQLYadyLcDIovzRFYZaCZZZYdYYpZDZYjGTTzjemgkm30p
0v3E1mxuZDDDjuPvitppwU1Pzjj00iqtd9mzjeImDiBk0D02uUUnShjZhluuh5NpNpLUW-
GpRelzw2zZhV0o1y6hhOZD2XjyyvmQshKduEu5|.vX-
gYi2GjwqBbszOmBnUd1lhJJZdJ9djTXLYMOOOOQM8MQc4QYQwwZQvIIiwOOIMZIMRNNNNQ1lNJMNyI8D
IDRdJNRhnFDXITlNummGuOemIOVnCIrrYI_LIddIIII2tUGIIbDb9E5bqpwRJjmQeuzUJtRzn5CO.z_2
YDNOWzIYIIYIgEMArK4mc_INJQMQUEU5V3_vKnw0w9wwdOz1ku9hQQ9RFiipyFl112kU0oYQccceeLM-
A7VVOjNmNgMYMBTIUkJ5L1VhicNDyQ",
    "token_type": "Bearer",
    "expires_in": "28800",
    "scope": "MYIR.Services",
    "refresh_token": "cks4mqqxrt9|gdpsrندر74szzz2sk3p9zthhm7vyf5542636d78"
}
```

#### 4.4.3 Error Response.

See section 2.1.3.5 for a more detailed list of error responses, a typical JSON response, HTTP Status code 401:

```
{
  "error": "invalid_client",
  "error_description": "The provided secret or assertion are not valid for this client. "
}
```

#### 4.5 Introspection token request (Validate)

Service consumer to IR. Refer to section 2.1.5

#### 4.5.1 Introspection token request

As a service provider you need to send a HTTP POST request to OAuth Server with Access Token OR Refresh Token to validate your client credentials.

To validate as Access Token set the form field "token\_type\_hint" to "access\_token" and to validate Refresh Token set the "token type hint" to "refresh token".

URL format:

<https://test5.services.ird.govt.nz/gateway3/oauth/introspect>

### Request headers:

Authorization: Basic VGVzdDMwMjA2NDkyOk9hdXRoMklsSU2VjcmV0dA==  
Content-Type: application/x-www-form-urlencoded  
Content-Length=972

### Request body:

[illegible]



```
oAGb0zPMPd0r_OnnUY2OH3CcVynskqcynIStMF12Ke95NcJHvd4Y6LHjw0bglonFg4sDVbAjs9La6SFs
0GclHUvQltCt3RhGovb4zYtoGD-4wTdJct_6oQz1UUxDCwJoAh_RuUbXGU1U1ifYCzw
&token_type_hint=access_token
```

#### 4.5.2 Introspection token reply (Validate)

A sample response payload HTTP Status 200:

```
{
  "active": true,
  "client_id": "nztaxSoftware",
  "username": "myIRusername",
  "scope": "MYIR.Services",
  "sub": "17acd64d-4fa3-4f41-a27d-ef18da156c28",
  "exp": 1656496794,
  "iat": 1656467994
}
```

#### 4.5.3 Introspection token Invalid Token Response.

A sample response payload HTTP Status 200 when the token is not valid:

```
{
  "active": false
}
```

### 4.6 Revocation token request (Revoke)

Service consumer to IR. Refer to section 2.1.6.

#### 4.6.1 Revocation token request (Revoke)

At this step as a service provider, you need to send a HTTP POST request to OAuth Server with Access Token OR Refresh Token to revoke your client credentials.

To revoke an access token set the form field "token\_type\_hint" to "access\_token" and to revoke refresh token set the "token\_type\_hint" to "refresh\_token".

URL format:

<https://test5.services.ird.govt.nz/gateway3/oauth/revoke>

Request headers:

```
Authorization: Basic VGVzdDMwMjA2NDkyOk9hdXRoMk1SU2VjcmV0dA==
Content-Type: application/x-www-form-urlencoded
Content-Length: 973
```

Request body:

```
token=eyJhbGciOiJSUzUxMiIsInR5cCI6ImlmF0LUpXVCIsImtpZCI6IjM5MEQ5QjYzNzBBMDE4Q0QwOU
RDRDVGQTA5MzMzOUVERTIzQUQ1RUQifQ.eyJyYmYiOiJlY2NTY0Njc2OTQsImp0aSI6IjI3Y2IwMDZmLTJ
iZDEtNGVmZi04MjQzLTIlYmQyNWFnNzcyZCIsImVudCI6Imh0dHBzOlwvXC90ZXN0MS5zZXJ2aWNlcy5
pcmQuZ292dC5uelwvZ2F0ZXdhcTNC129hdXR0XC8iLCJpYXQiOiJlY2NTY0Njc2OTQsInN1YiI6IjE3YWN
kNjRkLTRmYTMTNGY0MS1hMjQzLTIlYmQyNWFnNzcyZCIsImVudCI6Imh0dHBzOlwvXC90ZXN0MS5zZXJ2aWNlcy5
zY29wZSI6IklzSVIuU2VydmljZXM1LCJjbGllbnRpZCI6ImZhc3QiLCJhdWQiOiJodHRwczpcL1wvdGV
zdDEuc2VydmljZXMuaXJkLmdvdnQubncpL2dhZGV3YXkzXC9vYXV0aFwvIiwiaWF0IjE3YWNkNjRkLTRmYTMTNGY0MS1hMjQzLTIlYmQyNWFnNzcyZCIsImVudCI6Imh0dHBzOlwvXC90ZXN0MS5zZXJ2aWNlcy5
e5CEn-aHjBnVlpilnSik3YiEnsFFoRkKL3wEFLKuIiCAJtTP_PrFw0whuc59nYP3-vs96-
1W6I8xGGaiYSHWgs3HGfaSxfabXKaGEfPgqOd9AkR9u5IkSCzjD7z9CCxhUdt_Mtf0d3QOTGOILJb0CI
```

```
oAGb0zPMPd0r_OnnUY2OH3CcVynskqcynIStMF12Ke95NcJHvd4Y6LHjw0bglonFg4sDVbAjs9La6SFs
0Gc1HUvQltCt3RhGovb4zYtoGD-4wTdJct_6oQz1UUxDCwJoAh_RuUbXGU1U1ifYCzw
&token_type_hint=access_token
```

## 5 Appendix B – Glossary

Term	Meaning
Abbreviation/Term	Description
<b>Client Application</b>	<p>A Client Application is an operating instance of Software that is deployed in one or more sites.</p> <p>A number of deployment patterns are possible:</p> <ul style="list-style-type: none"> <li>• A single cloud-based instance with multiple tenants and online users,</li> <li>• An on-premise instance (e.g. an organisation's payroll system)</li> <li>• A desktop application with an online user.</li> </ul>
<b>Customer</b>	<p>A Customer is the party who is a taxpayer or a participant in the social policy products that are operated by Inland Revenue. The Customer might be a person (an "individual") or a non-individual entity such as a company, trust, society etc.</p> <p>Practically all of the service interactions with Inland Revenue are about a Customer (e.g. their returns, accounts, entitlements etc) even though these interactions might be undertaken by an Intermediary on their behalf.</p>
<b>Intermediary</b>	<p>A party who interacts with Inland Revenue on behalf of a Customer. Inland revenue's Customer is a Client of the Intermediary. There are several types of Intermediary including Tax Agents, PTSIs, PAYE Intermediaries etc.</p>
<b>Mutual authentication</b>	<p>Refers to two parties authenticating each other at the same time, being a default mode of authentication in some protocols (e.g. SSH) and optional in other (TLS).</p> <p>Mutual TLS can be referred to as mTLS</p>
<b>OAuth 2.0</b>	<p>OAuth 2.0 is an industry-standard protocol for authorization</p>
<b>Native app</b>	<p>An application that is installed by the user to their device, as distinct from a web app with a browser-based user interface to a back-end application.</p>
<b>Protected Service</b>	<p>A general term for the business-related web services that are accessed once authentication has occurred (e.g. the Return Service, the Intermediation Service, the Correspondence Service). This document describes the mechanisms that are used to authenticate access to Protected Services.</p>
<b>Resource Owner</b>	<p>Legal owner of the data or resources user in message exchanges with Inland Revenue</p>
<b>SFTP</b>	<p>Secure File Transport Protocol</p>

<b>Software</b>	This is the computer software that contains interfaces to (consume) the services that Inland Revenue exposes. Software is developed and maintained by a Software Developer and subsequently deployed as one or more Client Applications.
<b>Software Developer</b>	The person or people who design, implement and test Software. This build pack and the resources to which it refers are primarily focused on the needs of Software Developers. They might be commercial vendors of software or an in-house developer of software.
<b>Service Provider</b>	Entity running an application communicating with Inland Revenue
<b>PKCE</b>	PKCE is an extension to the Authorization Code flow to prevent several attacks and to be able to securely perform the OAuth exchange from public clients. PKCE (RFC 7636)
<b>TLS 1.2</b>	A cryptographic protocol that provides communications security over a computer network. Version 1.2 is mandated in most cases.
<b>WS-Security</b>	An extension to SOAP to apply security to Web Services. An OASIS Web service specification
<b>X.509 Certificate</b>	A digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate.

---

## 6 Appendix C—Creating a signing certificate and public/private key pair

Additional resources discussing JWT can be found at <https://jwt.io> and [ietf advice on jwt](#).

A self-signed Certificate can be created using openssl by:

Step 1: Generate a Private Key. ...

Step 2: Generate a CSR (Certificate SigningRequest) ...

Step 3: Generating a Self-Signed Certificate. ...

Step 4: Convert the CRT to PEM format. ...

Step 5: Configure Reporter to use the server.pem and private key.

### Linux (need root access):

```
# generate a public/private key pair
```

```
openssl genrsa -out jwtRS256.key 2048
```

```
# extract a RSA public Key
```

```
openssl rsa -in jwtRS256.key -pubout -outform PEM -out jwtRS256.key.pub
```

```
# generate the signature
```

```
echo -n
```

```
'ew0KCSJhbGciOiAiUmlrNTYiLCANCgkidHlwIjogIkpXVCIsIA0KCSJraWQiOiAiTTJNIG0KfQ.ew0KIC  
Aic3ViIjogIldpZ2V0IENvbXBhbnkiLA0KICAiaXNzIjogInd3dy53aWdldGNvbXBhbnkuY29tIiwNCiAg  
InN0YXJ0TG9nb24iOiAibXlJUndlYmxvZ2luVXNlciIsDQogICJpYXQiOiAxNTYwODk0NzMzLA0KICAi  
ZXhwIjogMTU2MDkwNDczMw0KfQ' | openssl dgst -sha256 -sign jwtRS256.key -binary |  
openssl base64 -A | tr -- '+/=' '-_'
```

### Windows:

```
# generate a public/private key pair
```

```
openssl genrsa -out jwtRS256.key 2048
```

```
# creates a RSA public Key
```

```
openssl rsa -in jwtRS256.key -pubout -outform PEM -out jwtRS256.key.pub
```

```
# generate the signature
```

```
echo -n
```

```
'ew0KCSJhbGciOiAiUmlrNTYiLCANCgkidHlwIjogIkpXVCIsIA0KCSJraWQiOiAiTTJNIG0KfQ.ew0KIC  
Aic3ViIjogIldpZ2V0IENvbXBhbnkiLA0KICAiaXNzIjogInd3dy53aWdldGNvbXBhbnkuY29tIiwNCiAg  
InN0YXJ0TG9nb24iOiAibXlJUndlYmxvZ2luVXNlciIsDQogICJpYXQiOiAxNTYwODk0NzMzLA0KICAi  
ZXhwIjogMTU2MDkwNDczMw0KfQ' | openssl dgst -sha256 -sign jwtRS256.key -binary |  
openssl base64 -A
```

The base64 signature needs to be made base64url encoded, which requires character substitution:

- '+' translates to '-'
- '/' translates to '\_'
- '=' translates to ' ' (space)

JWT Token:

```
ew0KCSJhbGciOiAiUmlhbnRlbnRlcANcGkidHlwIjogIkpXVCIsIA0KCSJraWQiOiAiTTJNIG0KfQ.ew0KIC
Aic3ViIjogIldpZ2V0IENvbXBhbnkiLA0KICAiaXNzIjogInd3dy53aWdldGNvbXBhbnkuY29tIiwNCiAg
InN0YXJ0TG9nb24iOiAibXlJUndlYm9mZ2luVXNlciIsDQogICJpYXQ1OiAxNTYwODk0NzZmLA0KICAi
ZXhwIjogMTU2MDkwNDczMw0KfQ.R4ht3N7jsMSRaeXAVSHtysNeymBexO-
0okop79EYVFqgvOZduvChyMIXkxSX_7HhQNKcJhtdOL64e0EPUiEIBpqS7GpA9cf8p_ktowej6ge
kjeVPSNHoDokYPbSi4y8Nb1OdSdr1ECKuSjsDvekc6vYN7f0j4Rrn3sr7Y-
0hpWEQgCiTcbUIfEHGm1nQHx9mFxDJYgyG4XDQXB1rF-LAIL_omDWGoRBBVyXSQk-
zb1uH08yhJv11j955Tz4GQD86DVvmgZPKoftPG73BGoyRzWaYH9vCi8AIZTRnAEUqwWvrw81MU
up1V6775L0cy830n0ka3tLSZ_SnbctDSiA
```

-----BEGIN PUBLIC KEY-----

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjLIJZI5ePxnDzoPChQ1w
2BINZySbPsvUC/4FgGmNmTH6s+V7YYpCqyIwQ1cLrsxvMdwKAXYslaVMhxbIRk3q
Xb5qWUv2Nb4dEMFBsWxqAPzbCuXAw2jlizyRSsqjotEcfxtYgjMOTCX41y1JpaUe
/tU+zKHBtCTwKvT05cFiXWAXbkyMDGg4yk79GumT5E1qC8f3eOMZhTcoiT599rMb
P+B3f9ka2W3g1jkaOiPiZ5sQRO93VggamjBXIR/I5JC3ljJpu86uwJuWITSJ1PE3
wZPxcNgb1Od60uc80ye1O1WTV0Wz+xYo2OnqAcBsoHti4f3o6A1SL7RCIUtIdIfE
VQIDAQAB
```

-----END PUBLIC KEY-----

Encoded

ew0KCSJhbGciOiAiUmlhbnRlbnRlcANcGkidHlwIjogIkpXVCIsIA0KCSJraWQiOiAiTTJNIG0KfQ.ew0KICAic3ViIjogIldpZ2V0IENvbXBhbnkiLA0KICAiaXNzIjogInd3dy53aWdldGNvbXBhbnkuY29tIiwNCiAgInN0YXJ0TG9nb24iOiAibXlJUndlYm9mZ2luVXNlciIsDQogICJpYXQ1OiAxNTYwODk0NzZmLA0KICAiZXhwIjogMTU2MDkwNDczMw0KfQ.R4ht3N7jsMSRaeXAVSHtysNeymBexO-0okop79EYVFqgvOZduvChyMIXkxSX\_7HhQNKcJhtdOL64e0EPUiEIBpqS7GpA9cf8p\_ktowej6gekjeVPSNHoDokYPbSi4y8Nb1OdSdr1ECKuSjsDvekc6vYN7f0j4Rrn3sr7Y-0hpWEQgCiTcbUIfEHGm1nQHx9mFxDJYgyG4XDQXB1rF-LAIL\_omDWGoRBBVyXSQk-zb1uH08yhJv11j955Tz4GQD86DVvmgZPKoftPG73BGoyRzWaYH9vCi8AIZTRnAEUqwWvrw81MUup1V6775L0cy830n0ka3tLSZ\_SnbctDSiA

Decoded

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "M2M"
}
```

PAYLOAD: DATA

```
{
  "sub": "Niget Company",
  "iss": "www.wigetcompany.com",
  "startLogin": "myIRwebloginUser",
  "iat": 156894733,
  "exp": 156894733
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  [UtiIfIE
  VQIDAQAB
  -----END PUBLIC KEY-----]
)
```

Private Key. Enter it in plain text only if you want to generate a new token. The key never leaves your browser.

Signature Verified

SHARE JWT

## 7 Appendix D – Deprecated, Aged and Changing Standards

Note that Inland Revenue reserves the right to upgrade cryptographic standards in response to factors such as external threats, industry standards and legislative change. IR's partners will be consulted, via IR's relationship managers, about changes of this nature.

The following standards are deprecated, have been superseded or are likely to be changed.

Standard	Description
<b>TLS 1.0</b> <b>TLS 1.1</b> <b>SSL (all versions)</b>	All versions of TLS prior to v1.2 and all versions of SSL have been compromised. No new implementations are permitted, and all existing implementations must be upgraded.  Note that TLS1.3 is now an industry standard but is not yet widely adopted (to do so requires upgrades to perimeter security devices and software). When TLS1.3 does becomes sufficiently widely adopted Inland Revenue will upgrade to it. Where practical external software partners should anticipate upgrading to TLS1.3.
<b>SHA-1</b>	No new implementations are permitted, and all existing implementations must be upgraded to at least a SHA-2 compliant algorithm.
<b>OAuth 1</b>	Inland Revenues does not and will not support OAuth 1.
<b>RSA</b>	The RSA suite of signing algorithms are still supported but use of them is discouraged in favour of the ecliptic curve signing algorithms. Refer to section 2.4 for the specific algorithms.
<b>Token Expiry</b>	The current settings for the OAuth access and refresh token's expiry are a trade-off between security and convenience. Refer to section 2.1 for the current settings of these parameters. IR reserves the right to change these values in response to security threats. External partners are therefore advised to parameterise these configuration settings.

## 8 Appendix E—Change log

Date of Change	Document Section	Description
<b>15/09/2017</b>	2.1.4 2.1.3.2	Remove reference to Refresh and Revoke token until further discussions and agreement with Service Providers  Amend Incorrect Scope value from "GWS" to "MYIR.Services"
<b>27/09/2017</b>	Multiple Appendix A – Sample payloads	Token time-out updated to 8 hours. Minor wording changes for ease of reading. Corrections to sample payloads where symbols had not correctly translated to word format e.g. https%3A%2F%2F to https://
<b>11/12/2017</b>	2.1.4 2.1.5 3.4 3.5	Added Refresh and Revoke token details and examples in the appendix.  Minor update to Service structure.
<b>23/01/2018</b>	2.1.4	Commentary added for refresh token validity
<b>05/02/2018</b>	2.2	Further detail for Service Providers offering desktop solutions regarding OAuth for Native apps.
<b>22/02/2018</b>	New section 2.1.3.4  New section 2.1.6  Use of 'State' in transaction calls Section 2.2	Insert new section describing the user of the consent screen  Insert new section describing the Validate Token Service  Security recommendation based upon RFC 6749  RFC 8252 Section 7.3 IRD unable to allow any port for return uri
<b>22/08/2018</b>	Add to page 10	Add revoke consent end points to assist with testing the authorise consent page flow as this is only displayed first time a user logs in.
<b>25/10/2018</b>	2.1.8 Endpoints  Amend section 2.1.3.2 to include logoff user parameter	Add the s.services test environment endpoint.  Include the logout parameter in the Request Authorisation Code call to force the logging off of any existing logged in myIR user. This will force the logon screen to be redisplayed for a new myIR user logon if desired
<b>28/05/2019</b>	Multiple	Amend end point references to TEST4.services~

<b>1/06/2019</b>	P 14 Table	Minor alts to make http 401 error visible STATE parameter limit added.
<b>10/06/2019</b>	Creation of M2M credential section	Description of JWT for use as an M2M credential. Includes a description of the signing certificate production. Added Appendix D.
<b>23/06/2020</b>	StartLogon can be optional	Amend StartLogon value in table for Client Signed JWT to be optional
<b>09/09/2020</b>	Inland Revenue upgrade of OAuth system	Inland Revenue updated the backend XIAMS system for OAuth Examples updated with minor changes to order of fields returned. Section 2.1.7 – Refresh tokens do not expire.
<b>23/12/2020</b>	2.1.3-5	Update to OAuth2.0 API error messages. Includes a new HTTP status codes, error descriptions and advises clients on remedial action.
<b>07/07/2022</b>		Release candidate build pack to introduce the new START IAM replacing the ORACLE xIAMS.
<b>05/10/2022</b>		Major Release of the new START IAM OAuth Service
<b>21/10/2022</b>	2.13.6 2.1.4 2.1.5 2.1.6	Content-Length is a required in the HTTP Header
<b>31/01/2023</b>	2.14	Detail explanation of the rotation of the family token set being revoked when refresh token reused.
<b>19/09/2023</b>	2.1.9	Removed reference to the old Oracle URL endpoints as it no longer applicable.
<b>24/11/2023</b>	3	New selection about TLS supported cipher for TLS 1.2 and TLS 1.3