Cabinet Office

OIX OPEN IDENTITY EXCHANGE

A TECHNICAL DESIGN FOR A BLUE BADGE DIGITAL SERVICE

The findings of an Alpha Project involving

Warwickshire County Council

GDS

DWP

innovate identity
experts in id

verizon

northgate PUBLIC SERVICES

mydex

By Steven Gevers (Verizon) and
Rob Laurence (Innovate Identity)

Introduction

Table of Contents

This technical paper should be read in conjunction with the white paper: *Towards an Architecture for a Digital Blue Badge Service*[1]. It describes the technical solution that was designed and built for an attribute exchange hub, as part of an OIX Alpha project, to support the digital delivery of a complex local government service.

The project involved a collaboration between Government Digital Service, Department for Work and Pensions, Warwickshire County Council, Mydex and Verizon to design an attribute exchange hub. The hub was built by Verizon with Warwickshire County Council building the relying party gateway to the hub. The attribute provider components were built by Verizon.

The proposed design was reviewed by the OIX Industry Working Group on Attribute Exchange who were able to provide an independent assessment of the approach taken, functionality incorporated and design resulting.

In this paper we set out the journey from design concept to delivery of a working attribute exchange system to satisfy the principal technical aim of the Alpha project – "to design and build a technical solution for attribute exchange, underpinned by the GOV.UK Verify service".

The technical solution for attribute exchange forms part of a generic platform intended to deliver services in local government efficiently and cost-effectively. The use case behind this Alpha project was the Blue Badge (disabled parking badge) application for citizens who qualify by means of an automatic eligibility.

[1] See http://oixuk.org/?page_id=444

Design considerations

The starting point for the technical design for the Alpha attribute exchange system was taken from the findings of the preceding Discovery project.

These findings included

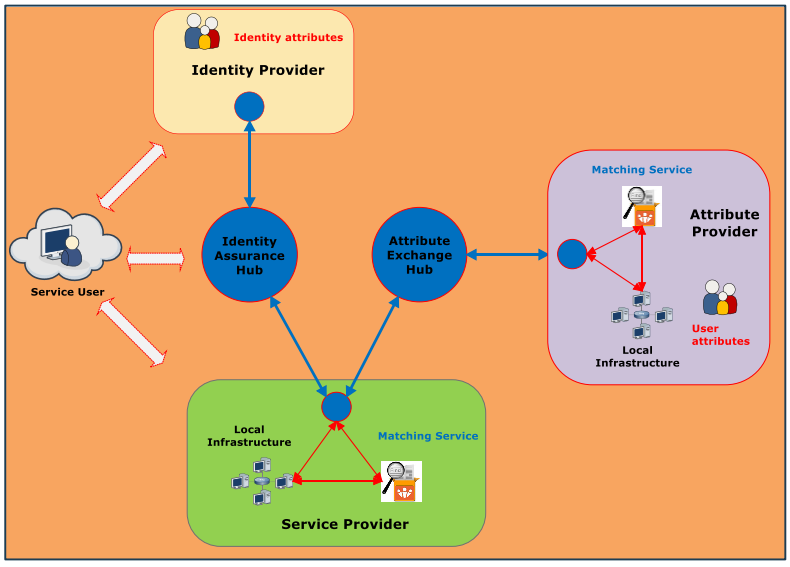
(a) user research
(b) design principles
(c) options for the high-level technical architecture

The Discovery project set out 3 potential designs for attribute exchange:

1. A combined identity assurance and attribute exchange hub
2. Separate identity assurance and attribute exchange hubs with attributes passing through the attribute exchange hub
3. Separate identity assurance and attribute exchange hubs with attributes passing directly from the attribute provider to the service provider (relying party)

The project team designed the attribute exchange hub based on option 2 (shown in the schematic below). This was selected for a number of reasons:

- identity assurance has already been designed and developed as a common capability within the government platform (ie GOV.UK Verify)
- identity assurance and attribute exchange can be treated as separate "services", each simpler in its own right and each able to develop at its own speed
- sending all of the messaging via the hub, rather than point to point between relying parties and attribute providers, simplifies on-boarding, and provides a consistent point for logging, auditing and billing. It better meets a number of the design principles established in the Discovery project



High-level technical architecture selected for Alpha project

Much care was taken by the project team to design the attribute exchange system in such a way that it forms the basis of a platform that can be developed and enhanced to include future requirements as envisaged by the project team. These were derived from the design principles, future potential needs, privacy guidance and potential technical constraints, as part of the design process.

The following table describes features of attribute exchange that were addressed as part of the design or were taken into consideration and recognised as being part of a future vision to develop attribute exchange ecosystems.

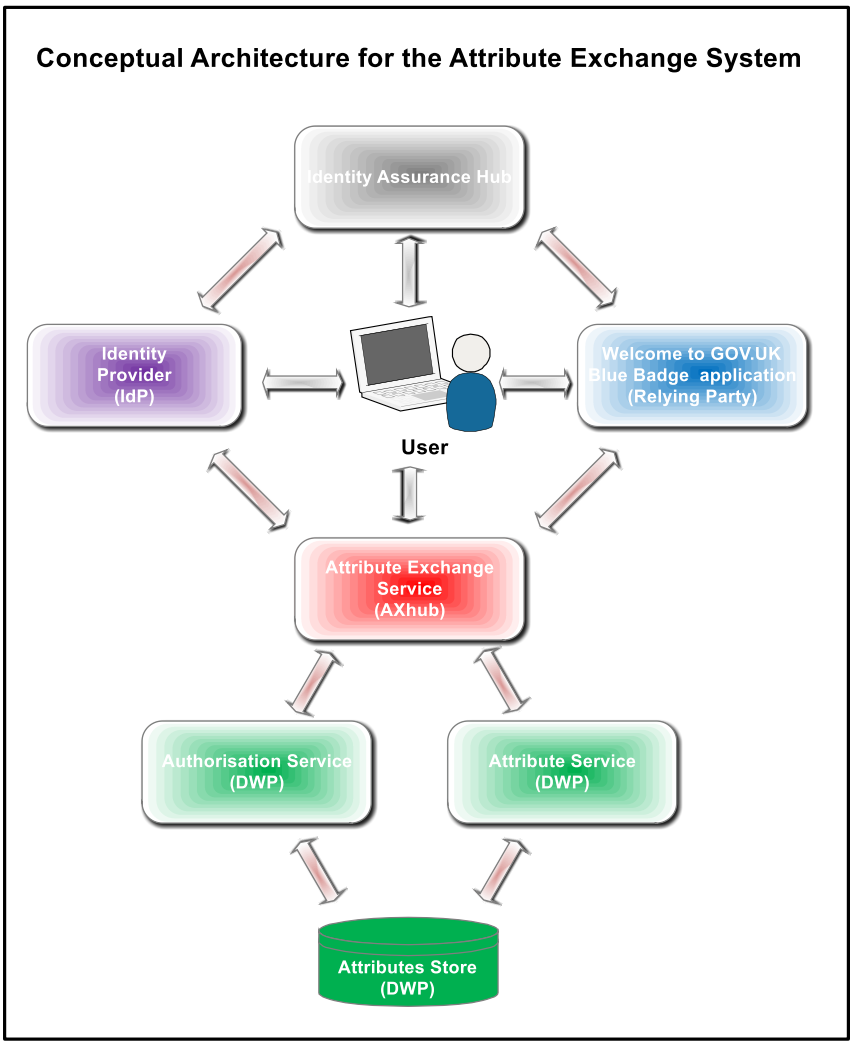
Feature	Description
Attribute data dictionary	A standardised approach to defining attributes, their sources, recency and associated levels of assurance. This wasn't required for the purposes of the Alpha but was viewed as an essential requirement within a future live environment.
Dynamic and static attribute exchange	Dynamic attribute exchange is the state where the relying party does not know who the attribute provider is. The decision on where to source the attribute from is made within the attribute exchange hub. Conversely, static attribute exchange is the state where the relying party does know who the attribute provider is. Whether it is dynamic or static is determined by the configuration in the attribute exchange hub. In this Alpha project, only static attribute exchange was supported.
Multiple consent	The user journey within the relying party digital service may need to access more than one attribute provider. To minimise interruption within the user journey flow, permission may be obtained at a single point by the relying party from the end-user to access all attribute providers.
Open source software and protocols	An ever increasing number of authorisation protocols are emerging, each with their own strengths. For the purposes of the Alpha project and the development of the attribute exchange hub, OAuth2 was selected as being the most suitable at this point in time. Attribute providers and relying parties should not be constrained by this. Therefore the hub should be capable of converting different protocols employed at the end points. The use of open source software, in this case ForgeRock authorisation software, removes the dependency on proprietary solutions and paves the way for collaboration across the public and private sectors to develop industry-wide solutions such as attribute exchange.

Privacy principles	The Privacy and Consumer Advisory Group (PCAG) Identity Assurance Principles have been written specifically for the identity assurance ecosystem. That said, the project team felt that these principles could equally apply to the attribute exchange ecosystem. The design of the user journey and technical solution was based on these principles. A formal piece of work is recommended to gain the insight of PCAG and to extend these privacy principles to attribute exchange.
Trust anchor	The trust anchor is the origination point of the trusted identity of the user. The user's digital identity credentials are passed from here to the attribute provider end-point in a "tamper-proof" form. In the target architecture the user's identity would be requested by the attribute exchange hub. The trust anchor would be the GOV.UK Verify identity assurance hub which would seek a re-authentication with the relevant identity provider specifically for the purposes of this attribute exchange. (In the current implementation of GOV.UK Verify, neither the identity assurance hub nor the identity providers can support this process; indeed the identity providers close down their sessions very soon after issuing the identity assertion. This means that for the attribute exchange hub to obtain the user's identity from the identity provider it would have to mimic some of the functionality of an identity assurance hub and ask the user to sign-in again with their identity provider. This is deemed to be an unacceptable user journey.) Within this Alpha project, the relying party is deemed to be the trust anchor. The relying party will forward the user's digital credentials and the attribute exchange authorisation service will check this. In practice the relying party's assertion of the user identity will include the equivalent of the Matching Data Set (MDS) and possibly other data to make matching of the user to a record at the attribute provider easier (eg a national insurance number to match within the DWP).
Web browser redirect (to the attribute exchange hub) – matching	The complexities of matching across end points is a challenge. The attribute provider will receive the relying party's identity assertion containing the matching data set (name, address, date of birth and gender) and possibly other data to assist location of the appropriate record. The attribute provider is expected to match this with the relevant user database entry. In order to improve the match the attribute provider may require some further information from the user. This requires the web browser to be redirected by the relying party to

	the attribute exchange hub. The hub can request information as determined by the attribute provider to support matching based on the status of the match and configuration in the hub. (This process of user mediation is allocated to the attribute exchange hub rather than the authorisation or attribute services to enable an attribute provider to be implemented using API calls and not require a web server interface. (This is consistent with the identity assurance hub design.)
Web browser redirect (to the attribute exchange hub) – permission	This design seeks the consent of the user, to the request for the attribute provider to release the attribute, at the relying party site. In the longer term the attribute exchange hub may seek permission on behalf of the attribute provider based on configuration at the hub. An example of this would be dynamic attribute exchange (see above).

Conceptual architecture

The conceptual architecture for the attribute exchange system that was derived for the Alpha project is shown below.



The architecture illustrates how the user is present and engaged throughout the transaction flow.

For the purposes of the Alpha project the identity assurance hub was built by Verizon, as were the components of the attribute exchange mechanism. The red box represents the service that is independent of the end points whereas the green boxes sit within the attribute provider's domain. The blue box was developed by Warwickshire County Council and represents the relying party's component.

Technical solution

The following section describes in detail the technical solution.

Description of components

The functional purpose and state of each of the components is described below.

Relying Party. This is the consumer of the attributes from the trusted source. It knows the identity of the user (asserted by GOV.UK Verify) and seeks permission from the user to obtain the attribute(s) it requires. In the Alpha project it was also the source of trusted identity.

Identity Provider. This is the source of identity assertions which are consumed by the attribute service. Ultimately the Verify provided identity will be reasserted by the Identity Provider to the attribute exchange hub. However, the current implementation of Verify does not support this. Within the Alpha project identity assertions for the attribute service were provided by the relying party (effectively acting as a lower trust identity provider for the attribute exchange).

In the diagram below, showing the technical design flows, ID and IDr in the Alpha were the same identity and the identity provider was, thereafter, not involved in the process flow.

Attribute Exchange Hub. This component handles the relying party's requests for attributes and provides the capability for user interaction during the process. It orchestrates calls to two attribute components – the attribute service and the authorisation service. It provides control over access to an attribute service by a relying party (perhaps technically ensuring that the exchange participants are those governed by the trust framework). It hosts capabilities to support user dialogues (for example, to assist matching or to seek permission from the user for attribute release). The hub handles security, registering certificates, encrypting and decrypting traffic and registering valid return URLs.

The hub could potentially support multiple protocols or act as an adapter within a protocol. As a central point in the information flow the hub could enable dynamic choice of attribute provider (ie dynamic attribute exchange) and provide billing services.

Authorisation Service. This confirms the legitimacy of the request from *the relying party* for *specific attributes* on behalf of the *asserted user identity* who has given their *stated permission*. The authorisation is granted for one time use only, based on the re-authenticated identity being still in session (as asserted by the identity provider in the target architecture case).

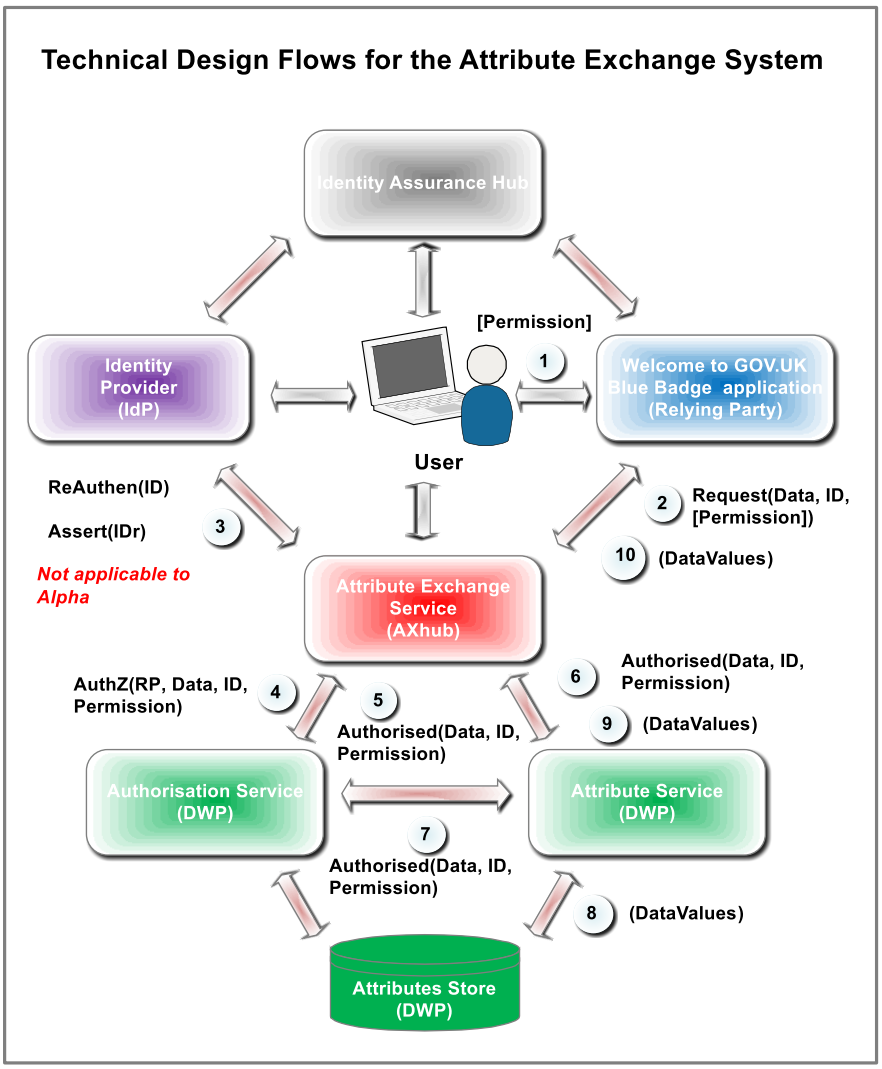
The authorisation service will *locate the relevant user record* (ie implement a matching service) so that the *whole* context is known and validated before the attribute service is called. It issues a token and token handle (represented in the technical design flows diagram as 'Authorised(Data, ID, Permission)') so that the attribute service can subsequently identify and verify the request.

This service could potentially request the hub to obtain further information to identify the user (through the web browser redirect function). It might also provide further checks on the user as required by the attribute provider; for example, instigating a watch list check. (This functionality was not implemented within the Alpha project).

Attribute Service. This component confirms that the authorisation is valid and then serves related attributes. The authorisation originates from the authorisation service and may be passed via the hub by value or by reference, but in either case the attribute service is closely related to its authorisation service.

Technical design flows

The attribute exchange process is described below, with each step numbered and cross-referenced between the diagram and the description.

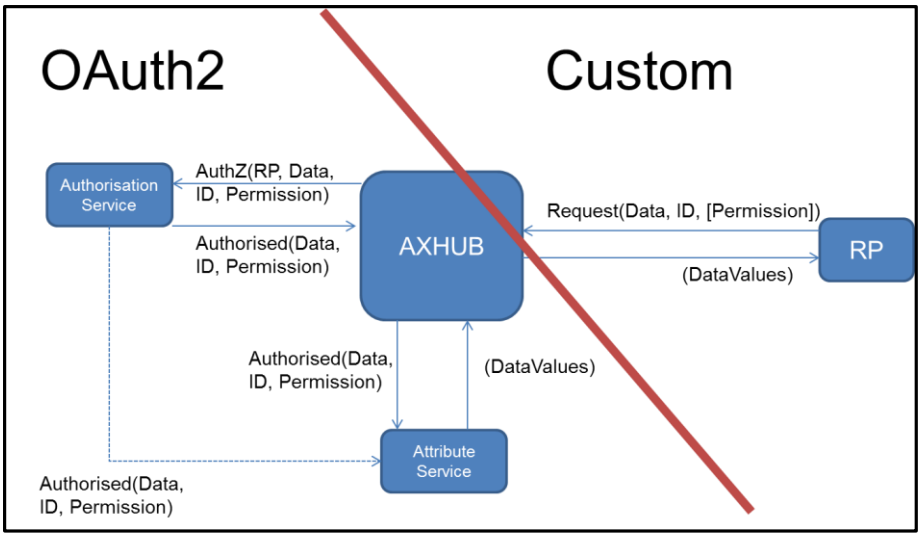


Step	Description
1	Prior to the attribute exchange process commencing the relying party will have obtained explicit permission from the user for the attribute(s) to be requested from the attribute provider. In giving this consent the user is made aware of where the attribute will be obtained from and for what purpose. The relying party will have also received an identity token from the identity assurance hub.
2	The relying party issues a request for each attribute citing the identity and confirmation that user permission has been obtained. The browser is redirected with the request to the attribute exchange hub.
3	The attribute exchange hub obtains a new identity assertion proving the user is still "in session". (This applies to the target architecture only, not the Alpha project).
4	The attribute request is handled by the attribute exchange hub. If the relying party is permitted to request this service from the relevant attribute provider, it forwards the request to the authorisation service.
5	The authorisation service verifies the request (as described above under the component description) and returns an authorisation token to the attribute exchange hub.
6	The attribute exchange hub issues a request to the attribute service for the attribute.
7	The attribute service checks that the request has been authorised by requesting confirmation from its authorisation service, either directly using the token handle (for example, by de-referencing the handle in an internal store) or by proving the legitimacy of the token itself (for example, by testing its integrity cryptographically).
8	The attribute service obtains the attribute from the data store.
9	The attribute service returns the attribute to the attribute exchange hub.
10	The attribute exchange hub returns the web browser to the relying party carrying the (signed) attribute.

Technical implementation

This section describes the technical implementation of the architecture within the Alpha project for the attribute exchange system described previously. It provides the basis of a viable technical approach for a "real world" implementation. Not all the technical details (e.g. the format of permissions, error flows) were worked out during the Alpha.

At a high level it is possible to separate the protocol used between the attribute exchange hub and the relying party on one side, and the authorisation service, attribute exchange hub and attribute service on the other side, as shown in the following diagram.

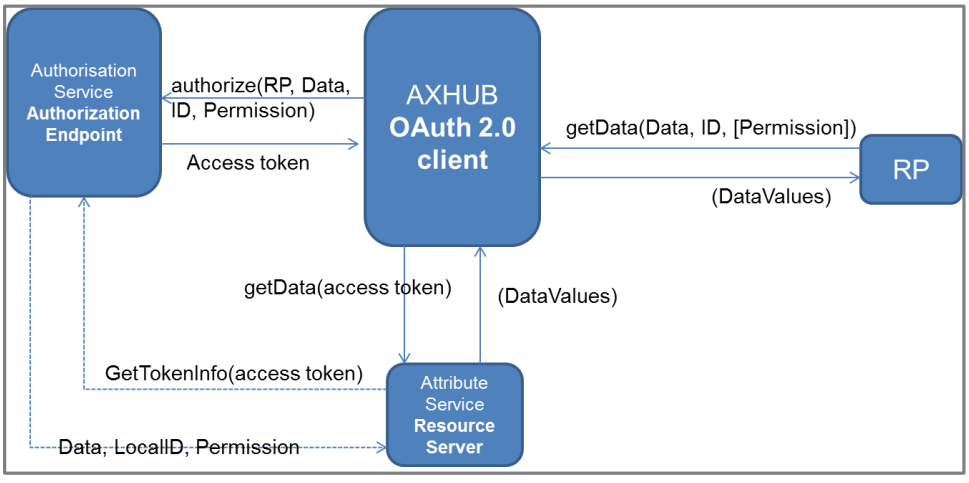


In this technical solution the relying party communicates using a custom call based on OAuth2 terminology.

In steps 5 and 6 described above, the attribute exchange hub has to retrieve a token or token handle from the authorisation service. Also, in some way, the attribute service needs to be able to verify this handle or token. In this solution, this was done using the OAuth2 protocol.

The attribute exchange hub acts as an OAuth2.0 client, the authorisation service acts as authorisation endpoint and the attribute service acts as resource server. First, the attribute exchange hub will request an access token from the authorisation service using the implicit grant flow. The authorisation service will return an access token for the attribute exchange hub. Next, the attribute exchange hub will use this access token to request attributes from the attribute service. For this functionality, the attribute service exposes a custom endpoint (getData) that requires clients to include an access token (bearer token) in the authorisation header. When the attribute service receives such a request, it will first verify the access token by calling the authorisation service using a getTokenInfo endpoint. The authorisation service will look up the access token and verify whether it is valid. Next it returns the requested attributes, the local identifier and the permission to the attribute service.

The following diagram shows this. In the next section each of the calls is described in more detail.



Detailed technical description of calls

This section provides a detailed description of each of the calls. Note that permissions are outside the scope of the Alpha. The calls below indicate when permissions should be communicated, but do not define the syntax or details of how they should be handled.

Call 1: GetData

This call is a custom call though some of the concepts are based on OAuth2. The relying party will redirect the user to the attribute exchange hub via a HTTP POST binding. It is necessary to use POST as the identity data of the user can be too large to fit in a URL itself.

URL	https://<<huburl>>/getData
type	POST
Post body	client_id=<<relyingparty_id>> redirect_uri=https%3A%2F%2Frelyingparty.com%2Fcb scope=bluebadge Permission={ < our standard for permission documentation > } ID={<ID>}_$Sig_{privatekey_RP}$_$Enc_{publickey_HUB}$

More details on the parameters

- **client_id:** the identifier of the relying party as known by the attribute exchange hub.
- **redirect_url:** the URL to which the attribute exchange hub must return the user's attributes. For security reasons it is important that the redirection URL's are registered in the hub during registration so it is not possible to modify them.
- **Scope:** the scope contains an overview of all data (ie attributes) that need to be returned to the relying party. For the Alpha, the requested scope will just be "bluebadge". Naming conventions for the attribute exchange ecosystem will have to be defined by the industry working group.
- **Permission:** the permissions provided by the user to the relying party. The format of the permissions is outside the scope of the Alpha.
- **ID:** this is an identity assertion for the authorisation service to identify the user. This can, for example, be a matching data set. Also, it could be enriched with other attributes if it helps matching. The ID will also require some security measures to be put in place. As the ID is passed via the browser, it should not be possible for the end user to read or modify the ID parameter. Also, it should not be possible to replay the parameter in a later transaction. For this reason, the ID will have the form of a JSON Web Token (JWT) (https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32), which is then signed using JSON Web Signature (JWS) (https://tools.ietf.org/html/draft-ietf-jose-json-web-signature) and thereafter encrypted using JSON Web Encryption (JWE) (https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption).

  The JWT specifications provide a number of "registered claims" for JWTs to contain security related information (e.g. expiry date). The following attributes MUST be provided in an ID that is used for attribute exchange: iss, exp, nbf, iat, jti. More information on these attributes can be found in Appendix A. Next to these attributes the user identity data needs to be included. An example of an ID message containing MDS:

```
{
 "nbf": 1428066438,
  "iss": "client_id",
  "exp": 1428067038,
  "iat": 1428066438,
  "jti": "8395de71-c92e-491a-ad65-bf5494cedc33",
  "MDS_firstname":"Patricia",
  "MDS_surname":"Naylor",
  "MDS_dateofbirth":"1959-11-01",
  "MDS_gender":"Female",
  "MDS_currentaddress": {
          "Line":"28",
          "Line":"High St",
          "PostCode":"BA133BN"
  }
}
```


This payload then needs to be signed by the relying party. Next, it is encrypted using the public key of the hub, so only the hub will be able to read the contents.

Verifications performed on the hub

The following checks need to be done before the hub is allowed to handle the request. If this is not the case, an error message is sent back to the relying party. The exact error messages are outside the scope of the Alpha.

- Verify whether the *client_id* represents a valid relying party that is allowed to request the *scopes*
- The attribute exchange hub needs to decrypt the MDS and verify
    - that the signature on the MDS corresponds to the *client_id*
    - that the iss corresponds to the *client_id*
    - Verify dates exp, nbf, iat
    - Verify that the value of jti was not used before
- Verify whether the *redirect_url* is registered as valid redirection url for the *client_id*
- Verify whether all required permissions are available (outside scope of the Alpha).

Call 2: Authorise

This call is based on OAuth2 implicit grant flow, loaded with some extra parameters. The hub will send a HTTP POST request in the backend to the authorisation service. This call is not passed via the browser.

URL	https://<<authorizationserviceurl>>/authorize
Type	POST
Post body	response_type=token client_id=<<relyingparty_id>> scope=bluebadge Permission={ < our standard for permission documentation > }

	ID={<ID>}_$Sig_{privatekey_RP}$_$Enc_{publickey_AuthorizationService}$

More details on the parameters

- **Response_type:** this value is always “token”; as defined by the OAuth2 specifications for the implicit grant flow
- **client_id:** this parameter is a copy of what was received by the attribute exchange hub in step 1
- **Scope:** this parameter is a copy of what was received by the attribute exchange hub in step 1
- **Permission:** this parameter is a copy of what was received by the attribute exchange hub in step 1
- **ID:** this parameter is a copy of what was received by the attribute exchange hub in step 1. Only the encryption changes, the attribute exchange hub first decrypts the parameter from step 1, and encrypts it using the public key of the authorisation service

Verifications performed on the authorisation service

The following checks need to be done before the authorisation service is allowed to handle the request. If this is not the case, an error message needs to be sent back. The exact error messages are outside the scope of this solution.

- Verify whether the *client_id* represents a valid relying party that is allowed to request the *scopes*
- The authorisation service needs to decrypt the MDS and verify
    - that the signature on the MDS corresponds to the *client_id*
    - that the iss corresponds to the *client_id*
    - Verify dates exp, nbf, iat
    - Verify that the value of jti was not used before
- Verify whether the *redirect_url* is registered as valid redirection url for the *client_id*
- Verify whether all required permissions are available (outside scope of this project)

Call 3: Authorise

In this call the authorisation service returns the access token according to the standards. The hub needs to extract the access token from the response.

Response code	302 (Redirect)
Redirect location	*http://<<attributeurl>>/cb#scope=bluebadge&token_type=Bearer&expires_in=59&access_token=4f57c734-2589-44aa-8473-58054da30afa*

Call 4: getData

In this call, the attribute exchange hub launches a request for a custom data endpoint. The endpoint is protected using OAuth2. As such, the access token received in the previous step is included in the authorization header.

URL	https://<<attributeprovider>>/getData
Type	GET

Headers	Authorization: Bearer <<accesstoken>>

Call 5: getTokenInfo
The attribute service now has to validate the access token and retrieve information on it.

URL	https://<<authorizationservice>> /tokeninfo?access_token=<<accesstoken>>
Type	GET

Call 6: Token info response
The authorisation service returns some information on the access token, including the requested scope.

Response code	200
Response body	{"scope":["localid","bluebadge"],"grant_type":"token","localid":"john.d" ,"token_type":"Bearer","expires_in":59,"access_token":"40185460-39b9-44fb-a18a-e5d3cd85059a"}

Call 7: Data response
The attribute provider can now look up the requested data and send it back to the attribute exchange hub. The way the data is returned is similar to how the identity assertion was passed in Call 1. Again, it is important that the user cannot read or modify the response. As such, again, a JWT will be built that will be signed and encrypted. As in Call 1, the JWT MUST contain attributes iss, exp, nbf, iat, jti. The data is signed by the attribute provider to ensure end-to-end integrity.

```
{
 "nbf": 1428066438,
  "iss": "attributeprovider_id",
  "exp": 1428067038,
  "iat": 1428066438,
  "jti": "8395de71-c92e-491a-ad65-bf5494cedc33",
  "bluebadge":"yes"
}
```

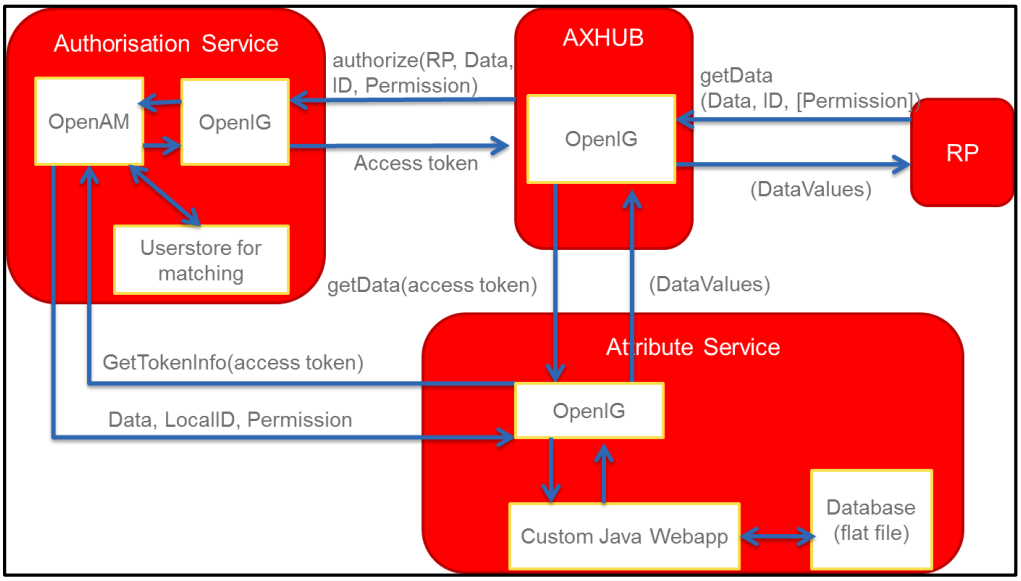
Firstly, this JWT is signed by the attribute provider. Next, it is encrypted using the public key of the attribute exchange hub. Finally, the encrypted value is then put in a JSON response as follows: {"attributes":"<<encrypted JWT>>"}

Call 8: Data response
Now the attribute exchange hub can forward the response received from Call 7. This time, the attribute exchange hub first needs to decrypt the response and encrypt it again, using the public key of the relying party. The response is sent back to the relying party via HTTP Post binding.

Build

The system build and test took place over a period of 4 weeks using open source software and the OAuth2 protocol.

The build was implemented using ForgeRock technology. Two ForgeRock products were used: OpenAM and OpenIG. OpenAM is the access management product. OpenAM is an identity provider solution with functionality to act as an OAuth2 provider. As such, it is used as the core component of the authorisation service. OpenIG (Open Identity Gateway) is a very flexible and extensible reverse proxy solution. It has flexible flow orchestration capabilities. As it is aimed at identity services it also provides out of the box protection for REST endpoints using OAuth2. Because of the flexibility of the product, it is used for different reasons in the attribute exchange hub, the authorisation service and the attribute service.



Attribute exchange hub setup

The role of the attribute exchange hub is to handle data requests. After receiving a request, it executes the following steps:

- decrypts the ID parameter
- performs verifications
- re-encrypts the ID parameter
- requests an authorisation (access token) from the authorisation service
- sends a request for attributes to the attribute service including the access token
- re-encrypts the response
- sends the response back to the relying party

Sending requests to the authorisation service and the attribute service was provided “out of the box” using the different request handlers provided by the product.

Implementation considerations

The implementation of the attribute exchange hub using the chosen product was straightforward. None of the components gave issues.

In a real life scenario, the solution would also have to include capabilities to show flexible screens, multiple attribute services, billing, auditing, and possibly handle multiple protocols. As these will require custom logic, it is expected that significant development would be required to build an attribute exchange hub.

Many different kinds of implementations can be imagined. The main role of the attribute exchange service is the orchestration of the attribute exchange flow. As such it is expected that, for example, existing Enterprise Service Bus (ESB) frameworks would provide a solid basis to start from.

Authorisation service setup
The authorisation service is mainly an OAuth2 authorisation endpoint. It receives a request for authorisation, validates the request, looks up the user in its data store, verifies whether it can grant authorisation and returns an access token.

In the Alpha, the core of the authorisation service is OpenAM. Out of the box, OpenAM provides an endpoint to first authenticate a user. It also provides an authorisation endpoint for authenticated users. However, the proposed protocol launches only one request for authorisation; this request includes the authentication information (the ID parameter). Therefore, OpenIG was added. OpenIG receives the authorisation requests from the attribute exchange hub. It first validates the request. Next, it prepares to authenticate the user through OpenAM by calling the authentication endpoint and sending the ID parameter (in the Alpha, only the givenname was used). OpenAM can then use this information to match it to the right local user. When the right user is found, OpenIG can launch a subsequent request to the authorisation endpoint using standard OAuth2.

Implementation considerations
The main responsibility of the authorisation service is to act as an OAuth2 authorisation service. This functionality can be built on OAuth2 libraries. However, there are also other OAuth2 identity provider products on the market that provide the required functionality.

In case an identity provider product is used for the implementation of an authorisation service, it is key to analyse the extensibility. The most important parts to be aware of are as follows.

- Many products require multiple calls to first authenticate the user and then to request the authorisation or access token. In the Alpha this was solved by adding the OpenIG component to orchestrate this process.
- The user needs to be authenticated using the ID parameter. The parameters need to be parsed, and be fed into a matching component to find the corresponding local account.

Attribute service setup
The attribute service is responsible to find user data based on an access token it received.

In the Alpha, the OpenIG component is configured to act as the OAuth2 resource server. It fetches information about the access token (the attributes to fetch and the local identifier) from the authorisation service. The call is then forwarded to an internal data service. For the Alpha,

the data service is implemented as a simple Java web service. It receives the local identifier from OpenIG, looks up the corresponding information in a flat file, and puts the information in a valid response.

Implementation considerations
The attribute service is in fact a simple REST endpoint protected by OAuth2. This functionality is offered out of the box by many products (e.g. ESB technologies) and vendors. As such, the best implementation in a real life scenario mostly depends on the technologies that are already in place.

- If an ESB solution is in place, it is expected that it can provide the functionality relatively easily
- If current technologies already have REST capabilities, several products exist that can protect the endpoint using OAuth2, comparable to the functionality OpenIG provides as described above
- If the technology is not REST or Oauth2 aware, products like OpenIG (several others exist), can be used to make the connection to the internal applications

Findings

During the course of the Alpha project considerable time and effort, and counselling of parties outside the immediate project team, was made to ensure the design reached was considered suitable and effective to meet the longer-term objectives of attribute exchange – to enable complex public AND private sector citizen and consumer services to be delivered digitally and more effectively.

The task of reaching a conceptual architecture was relatively straightforward, as a progression from the preceding Discovery project.

One area that involved much discussion was that of open source software and protocols. The project team recognised that different entities may wish to have choice over the protocol used and that also protocols were fast-developing and addressing new areas (for example, User Managed Access). The conceptual architecture allowed for this with the attribute exchange hub supporting a range of protocols that entities may wish to employ.

Whilst the Alpha was based on OAuth2, other protocols such as OpenID connect, SAML, UMA, etc, may be utilised in the solution as circumstances require.

The protocol and open source software for the Alpha was chosen as it is very straightforward and the subsequent ease of build was testimony to this.

All in all, no significant challenges or limitations were encountered in the delivery of the Alpha project technical solution.

Appendix A: JWT claims

For reference, this section provides the registered name claims as used in the JWTs exchanged during the attribute exchange process. This section is an exact copy of https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32#section-4.1.

4.1.1. "iss" (Issuer) Claim

The "iss" (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific. The "iss" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

4.1.2. "sub" (Subject) Claim

The "sub" (subject) claim identifies the principal that is the subject of the JWT. The Claims in a JWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique. The processing of this claim is generally application specific. The "sub" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

4.1.3. "aud" (Audience) Claim

The "aud" (audience) claim identifies the recipients that the JWT is intended for. Each principal intended to process the JWT MUST identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the "aud" claim when this claim is present, then the JWT MUST be rejected. In the general case, the "aud" value is an array of case-sensitive strings, each containing a StringOrURI value. In the special case when the JWT has one audience, the "aud" value MAY be a single case-sensitive string containing a StringOrURI value. The interpretation of audience values is generally application specific. Use of this claim is OPTIONAL.

4.1.4. "exp" (Expiration Time) Claim

The "exp" (expiration time) claim identifies the expiration time on or after which the JWT MUST NOT be accepted for processing. The processing of the "exp" claim requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

4.1.5. "nbf" (Not Before) Claim

The "nbf" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

4.1.6. "iat" (Issued At) Claim

The "iat" (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the JWT. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

4.1.7. "jti" (JWT ID) Claim

The "jti" (JWT ID) claim provides a unique identifier for the JWT. The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string. Use of this claim is OPTIONAL.