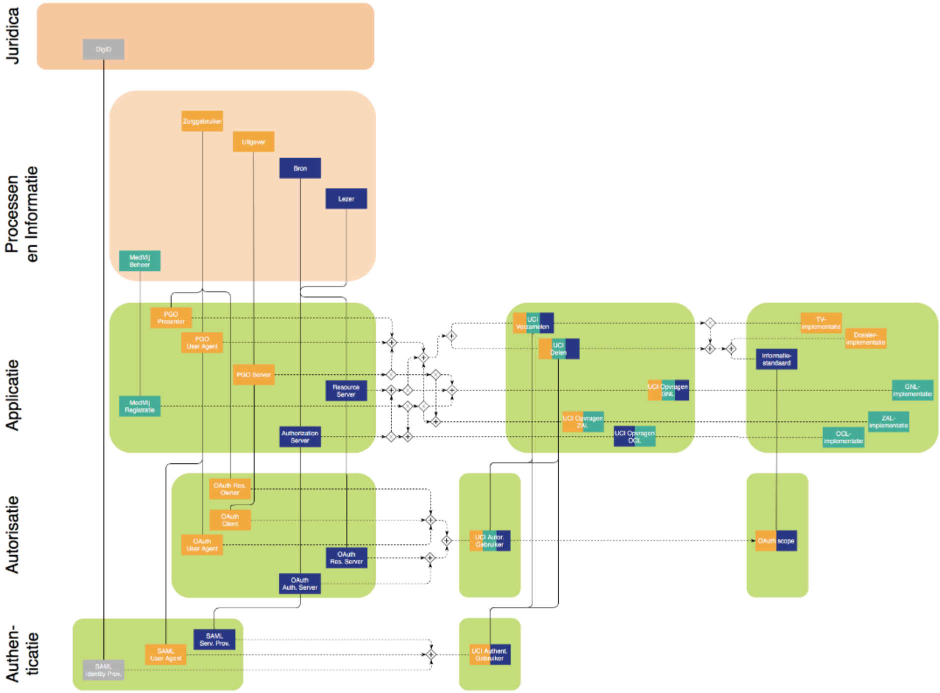
Application



Explanatory Notes

For an overview of all layers of the architecture, and for explanatory notes on the meaning of the symbols and lines, see the overview page.

The abbreviation:

- *CD* stands for *Consent Declaration;*
- *CPL* stands for *Care Providers List;*
- *OCL* stands for *OAuth Client List*;
- *DSNL* stands for *Date Service Names List*.

Roles

1. *Publisher* provides *Care User* with, in the context of the applicable *Service Provision Agreement*, an automated system for use, here called the: *PHE Server*. A single *Publisher* provides one or more *PHE Servers*, with each *PHE Server* being provided by a single *Publisher*.
2. *Source* provides, and *Reader* provides, an automated service, for the exchanging - on behalf of care providers - of health information with *PHE Server*, which consists of:

Authorisation Server and *Resource Server*. A single *Source and/or Reader* provide(s) one or more combinations of a single *Authorisation Server* and a single *Resource Server* and each combination of a single *Authorisation Server* and a single *Resource Server* is provided by a single *Source* and/or *Reader*.

3. *Care User* uses two automated roles for access to the functionality of *PHE Server* and *Authorisation Server*: *PHE Presenter* for the presentation of the functionality to *Care User* and *PHE User Agent* for the addressing of *PHE Server* and *Authorisation Server*.
4. *MedMij Management* makes an automated service available to all stakeholders , which is named here: *MedMij Registration*.
5. To authenticate the *Care User*, the relevant *Authorisation Server* (in this release of the MedMij Framework) will make use of *DigiD* as *SAML Identity provider*, in line with the SAML 2.0 interface of DigiD, whereby:
    1. the SAML role of *User Agent* is provided by the *PHE User Agent*;
    2. the SAML role of *Service Provider* is provided by the *Authorisation Server*;
    3. the SAML role of *Identity Provider* is accordingly provided by *DigiD*.
6. To authorise *PHE Server* to access the *Resource Server*, in the context of the functions *UC Compile* and *UC Share*, the relevant *PHE User Agent*, *PHE Server, Authorisation Server* and *Resource Server* will make use of OAuth 2.0, whereby Authorisation Code will be used as the grant type and whereby:
    1. the role of *OAuth User Agent* is provided by the *PHE User Agent*;
    2. the role of *OAuth Client* is provided by the *PHE Server*;
    3. the role of *OAuth Resource Server* is provided by the *Resource Server*;
    4. the role of *OAuth Authorisation Server* is provided by the *Authorisation Server.*
7. *MedMij data transfer* is defined as: all the data transfer in the context of any use case implementation in this layer or in the Network layer, directly between two different roles that are of the following four types of roles, namely:
    - first of all: *PHE Server*,
    - second, *PHE User Agent*,
    - third, Authorisation *Server* or *Resource Server* and
    - fourth, *MedMij Registration*,

   provided that:

    - in all cases these roles contain any *OAuth* roles that they respectively provide,
    - in all cases these roles exclude any *SAML* roles that they respectively provide, and

    - in all cases these roles, with regard to the use case implementations in the Network layer, include the Network roles that they function on.
8. All the *MedMij data transfer* - in so far as the *PHE User Agent*:
    - is involved in it - is called: *frontchannel data transfer*;
    - is not involved in it - constitutes the *backchannel data transfer*.

Explanatory Notes

Here, the roles of the Processes and Information layer are translated into roles on the application level. For details of the general basic principles regarding the numerical relationships between the roles, see the page Architecture and technical specifications.

In the individual's domain, three roles are distinguished between: the *PHE Presenter*, *PHE User Agent* and the *PHE Server*. This is necessary in order to be able to make the connection with authentication roles in line with OAuth. *PHE Presenter* and *PHE User Agent* are all front-end roles for the *PHE Server*, and can both be implemented in a browser for instance, but for a good binding to the *OAuth role* and *SAML* role, and to have good security measures, it is necessary to distinguish between these two roles. As elsewhere in the MedMij Framework, here too it is about roles, in other words about sets of responsibilities, and not about implementation components.

In the care provider's domain , it is not necessary to make such a distinction. Where an *Individual* is operationally involved in the information transfer - namely to have themselves identified and authenticated, and to have the transfer authorised - the *Care Provider* has completely represented themselves in operational terms by their service provider and their *Authorisation Server* and *Resource Server*. Even though in many cases the health information will ultimately be obtained from an underlying system, this is not an issue for the MedMij Framework. It is sufficient to place the ultimate responsibility (black box) with the *Authorisation Server* and *Resource Server*.

In line with options in the Process and Information Layer, these servers act on behalf of any and all underlying systems in the care provider's domain, such as xIS systems. This underlying complexity is a black box. It is possible that an individual xIS acts for both servers but then all responsibilities linked with these roles must have been filled in too, both the directly linked responsibilities (in the Application layer) and the indirectly linked responsibilities (in the layers above and below).

The choice made, in OAuth, to opt for the grant type Authorisation Code fits the typical software architecture found in the Individual's Domain for MedMij: access to a PHE service via components that are not under the control of the *OAuth Client* and that must be considered to be relatively unsafe. In this layer, in respect to this access we distinguish between two roles: the role *PHE Presenter*, who is responsible for the presentation of the functionality to the *Care User*, and the role *PHE User Agent*, who is responsible for the *PHE Server* and the *Authorisation Server*. It is the role *PHE User Agent* that is linked with the roles *OAuth User Agent* and *SAML User Agent*. In other words, the *PHE User Agent* ultimately addresses *DigiD* too, which is the *SAML Identity Provider*.

In the current MedMij Framework, the roles *Authorisation Server* and *Resource Server* work together in the same synchronous session. Their mutual relationship is a process link. In other words, they are orchestrated under a single process. However, roles in the MedMij Framework are groups of responsibilities, not implementation components. This means it is down to the *Care Provider's Service Provider* to make choices in his implementation and in his business model about whether to keep separate or actually combine these two roles. If the roles are separate then it is also certainly possible that a single *Authorisation Server* can work with multiple *Resource Servers* and that a single *Resource Server* can do business with multiple *Authorisation Servers*. However, together they must always demonstrate the behaviour that is required by the MedMij Framework. By the way, the OAuth specification also mentions this discretion (i.e. freedom) regarding implementation.

Because the session coordination in the Care Provider's Domain extends across the dividing line between *Authorisation Server* and *Resource Server*, an interface must be realised in which this session coordination is retained in the case of the separate implementation of *Authorisation Server* and *Resource Server*. In addition - if the relationship between *Authorisation Server* and *Resource Server* is not one-to-one - it must be ensured that the right two find each other for the communication about a specific access token.

Despite this discretion regarding implementation, the responsibilities in the MedMij Framework influence the implementation architecture in the Care Provider's Domain. The addressing in particular requires it to be the case that for a single combination of *Care Provider* and *Data Service* (and *system role respectively*) there can only be a single authorisation endpoint and a single token endpoint (and a single resource endpoint respectively). In addition, restrictions on the information content of authorisation codes and access tokens prevent the interface between *Authorisation Server* and *Resource Server* from being realised via the Individual's Domain. Apart from a few exceptions, that interface is an internal matter for the Care Provider's Domain. This serves principles P1 and P7 as well as that of minimisation, and thus that of privacy too. The most important exception is that the authorisation code and the access token may if desired contain an identification of the service that issued it. In this way, the (actual or intended) acceptor of the authorisation code or the access token can find the *Authorisation Server* where the validation of the authorisation code or the access token must take place.

Even if there is separate implementation, it is still a single *Care Provider's Service Provider* who is ultimately responsible in respect of MedMij for the joint behaviour of *Authorisation Server* and *Resource Server*. This means that the interoperability between *Authorisation Server* and *Resource Server* must come under the agreement that the relevant *Care Provider's Service Provider* establishes with any subcontractors, for example, if the *Care Provider's Service Provider* itself operates the *Resource Server* but contracts a subcontractor for the *Authorisation Server*. See also the explanatory notes under the Roles on the Network page for details of how *Nodes* are dealt with at a network level if a *Care Provider's Service Provider* uses subcontractors for *Authorisation Server* functionality, for example.

It is conceivable that a community of service providers in the Care Provider's Domain agree a framework alongside and in compliance with the MedMij Framework, in which the matter of the internal architecture of the Care Provider's Domain is addressed. In addition to the aforementioned separation, this could for instance include architectural choices about the availability test and receptiveness test.

The standards OAuth 2.0 en SAML 2.0 have different goals: OAuth for authorisation and SAML for authentication. Amongst other things, this ensures that the role structure is different. In OAuth there is a user (*Resource Owner*) that via his browser or app (*User Agent*) provides access for one application (*Client*) to another one (*Resource Server*), which last-named one has itself assisted for this by an *Authorisation Server*. In SAML there is a user that uses a browser or app (*User Agent*) to log in to a service (*Service Provider*), which has itself assisted for this by an *Identity Provider*.

All the same, there are important similarities between the ways in which they work.

- Both assume that the end-user presents himself via a relatively unsafe channel (the *User Agent*, the "front-channel"), while at the same time more sensitive information must be exchanged ("back-channel") that does not pass through this channel.
- In both cases, the *User Agent* must be led back and forth (redirect). In the case of OAuth, this is from *Client* to the *Authorisation Server* and back. In the case of SAML, this is from the *Service Provider* to the *Identity Provider* and back.
- In the case of both, the service provider (in the case of OAuth the *Authorisation Provider* and in the case of SAML the *Service Provider*) do not receive the desired information [in the case of OAuth the access token, in the case of SAML the user's identity and in the case of DigiD the BSN (citizen service number)] directly but via a retrieval certificate (in the case of OAuth the authorisation code, in the case of SAML the artefact). The retrieval certificate passes in front (via the *User Agent*), after which the desired information is retrieved behind with the retrieval certificate.

Article 7 delineates the *MedMij data transfer* in respect of the Network layer. All the *MedMij data transfer* is across domain borders. In addition, neither any transfer between *PHE Presenter* and *PHE User Agent* nor any transfer between *Authorisation Server* and *Resource Server* is part of *MedMij data transfer*. SAML data transfer is excluded, because the MedMij Framework cannot impose any requirements on DigiD. This delineation is also the prelude to Article 8. The distinction made therein between frontchannel transfer and backchannel transfer is necessary for the formulation of responsibilities about addressing (see Data and performance in UCI Compile and UCI Share) and security (see Network). Article 7 must take into consideration that there is also a use case implementation in the Network layer: *UCI Request WHL*.

Responsibilities

Explanatory Notes

The responsibilities in this layer and those in the processes and information layer have a similar structure. They are organised into chapters and sections as follows:

- File and consents
    - Use cases
    - Data Services
    - Authentication
    - Authorisation
- Lists
    - Care Providers List
    - OAuth Client List
    - Data Service Names List
- Security

For five of the six use cases (see the layer Processes and Information), a use case implementation (UCI) is prescribed in this (Application) layer. Implementation of the use case *Consult file* lies outside the scope of the MedMij Framework. So it is regarding the following five:

use case implementation	**Flowchart**
UCI Compile	with

UCI Share	with
UCI Request CPL	with
UCI Request OCL	with
UCI Request DSNL	with

File and consents

Use cases

1a. The above roles implement the use case *UC Compile* with the use case implementation *UCI Compile*. They use the relevant flowchart for this. The entire process is carried out synchronously.

1b. The above-named roles implement the use case *UC Share* with the use case implementation *UCI Share*. They use for this the relevant Flowchart. The entire process is carried out synchronously.

Explanatory Notes

In this release of the MedMij Framework, the use cases *UC Compile* (one-off compiling) and *UC Share* (one-off sharing) are the only ones in which health information is shared. Because the compiling and sharing are one-off in nature, authorisation and authentication can still be interwoven in the relevant flows. The user experience is best served by keeping the entire process synchronous.

Data Services

2. If a *Publisher* makes a certain *Data Service* available for his *Care Users* and to this end arranges delivery by a *Source* or *Reader* then the *PHE Server* of this *Publisher* and the *Authorisation Server* and *Resource Server* of this *Source* or *Reader* respectively will implement for this the use case implement that belongs to the *Data Service* and use the Information Standards that belong to the *Data Service* as *included* in the *Catalogue* .

Explanatory Notes

In this way it is guaranteed that the correct use case implementations and information standards are used.

Authentication

3. During the use case implementations *UCI Compile* and*UCI Share*, the *Authorisation Server,* for his role in these use case implementations and in his SAML role as *Service Provider*, has - immediately after the start of the OAuth-flow and before it asks the *Care User* for OAuth-authorisation - the *Care User* authenticated by DigiD, in line with the SAML 2.0 interface of DigiD.

Explanatory Notes

In accordance with Flowchart under 1. The care provider in the Care Provider's Domain and thus in the BSN domain is obliged - for the provision of data from a health file - to use the BSN to verify the identity of the individual. It follows from the Legal framework that for the time being, DigiD will be used for this purpose.

The MedMij Framework places the calling of DigiD in the OAuth-flow, under the operational responsibility of the *Authorisation Server*.

The last-named acts in this regard under the responsibility of individual *Care Providers*, because they are the reason why the *Individual* authenticates himself. Given DigiD's current status, this means that in this regard it is a special case that the *Individual* who is authenticating himself is not the direct user of the caller of DigiD (where they receive a user session) but an indirect user, namely through the intermediation of the*PHE Server*.

The direct interaction of the *Individual* with the *Authorisation Server* is intended to authorise the *PHE Server* to address the *Resource Server*. And this ultimately delivers the service. Traditionally however, DigiD has been set up for the authenticating (logging in and logging out) in user sessions with service providers. The MedMij context is more complex than that. When it comes to the future of DigiD, usage contexts such as those of MedMij are being considered too.

For the short term however, a number of DigiD connection requirements are raising issues about how they have to be interpreted in the MedMij context.

First of all, DigiD requires the end-user to retain the option of logging out “during the user session”. For the *UC(I) Compile* and *UC(I) Share* , this “user session” may be deemed to not extend beyond the *Authorisation Server*. After the *Individual* is no longer a “user” of the *Authorisation Server* but of the - irrelevant for DigiD - *PHE Server*, that in its turn has become the - and authorised to this end by the *Individual* - user of the *Resource Server*. During the session with the *Authorisation Server*, there is also only a single end-user interaction, namely in the case of the presentation of the consent / confirmation request. The rejection of this consent / confirmation by the *Individual* may be interpreted as logging out. If the user does nothing in the authorisation screen for fifteen minutes or if the user closes this screen then the session must be terminated. No further logout arrangements are necessary.

Secondly, DigiD lays down requirements for the “DigiD session data” and for the data derived from it. This raises the issue to which data this relates and whether the Authorisation code and the access token must be considered to be derived data. The reason for this requirement is that the data referred to is needed to log out again, especially in the case of single sign-on. In *UC(I) Compile* and *UC(I) Share* ; however, neither data delivered by DigiD nor data “derived” from it is used for logging out. This means this requirement is being complied with beforehand. If single sign-on becomes possible as well in MedMij, then this must naturally be reviewed.

Thirdly, DigiD requires that after authentication there is redirection to the same domain as the one from which DigiD was called. This is complied with by the MedMij Framework anyway, because this DigiD call and DigiD redirect are embedded in a comparable Oauth call and Oauth redirect. The situation is thus interlocked: the *PHE Server* calls the *Authorisation Server* , which calls DigiD, which redirects to the*Authorisation Server*, which follows the OAuth-flow, in which a redirect to the *PHE Server*takes place amongst other things. This nesting actually makes this more complex but this requirement does continue to be complied with.

In this way, the MedMij Framework fits with the current DigiD. If DigiD develops further (for example towards app-to-app data transfer) or if the MedMij Framework continues to develop (for example towards single sign-on) then this passing must be looked at again.

Authorisation

4. During the use case implementations *UCI Compile* and *UCI Share*, the *Authorisation Server* - as soon as the *Care User* has been authenticated as referred to under 3 - continues with the OAuth authorisation, in line with the standard OAuth 2.0.

Explanatory Notes

In accordance with the statutory obligation, *Care User* gives - in the *UC Compile* - his active consent to the *Care Provider.* In the *UC Share*, this requirement does not apply, but even so a confirmation is made at this moment by the *Care User*. The *PHE Presenter* presents a window in which the *Care User* can give this consent or confirmation respectively. Since the BSN cannot be used in the Individual's Domain, a substitute identification of the care user must be used. See responsibility 5.

5. In so far as the transfer between *PHE Server* and *Resource Server* , namely in the use case implementations *UCI Compile* and *UCI Share*, there is, in the control data, a data element from which the identity of the *Care User* can be derived, they use for this nothing else than the OAuth data that they had to exchange in their respective *OAuth Client* and *OAuth Resource Server* . *PHE Server*, *Authorisation Server* and *Resource Server* make properly secured arrangements where they can if necessary establish the identity of the *Care User* themselves.

Explanatory Notes

With a view to guaranteeing privacy and keeping the architecture of the MedMij Framework as simple as possible, it has been decided to keep the identifier for the *Care User* as meaningless as possible 'en route'. All meaning is linked on both sides by consulting internal registrations. Because the *PHE Server, Authorisation Server* and *Resource Server* process a OAuth flow together, they possess (after authentication of the *Care User*) tokens that represent the identity of the *Care User*, namely (first) the authorisation code and (later) the access token. Apart from these, no identifying data elements need to be or will be included in the transfer. The FHIR data element *PatientID* will *not* be used.

6. OAuth2.0 provides four types of authorisation grants, but the OAuth roles limit themselves to the Authorisation Code.

Explanatory Notes

This one type can be used to serve all the situations that occur in the MedMij Framework. In order to maximise the interoperability, MedMij opts to exclude the other three types.

7. The *OAuth Client* and *OAuth Resource Server* will only exchange tokens of the type Bearer Token, in accordance with RFC6750. To send the access token, the OAuth Client uses the method Authorisation Request Header Field, as described in section 2.1 of RFC6750.

Explanatory Notes

It is the OAuth standard that releases the (access) token type. Token types differ in the degree of confidence with which the *Resource Server* can provide the requested resources to the *Client* when the last-named submits the access token to the former. In its simplest form (Bearer Token), the *Resource Server* simply provides the related resources to each *Client* who submits a valid access token. This is done "to bearer", in the same way that a bank can cash a check to bearer. However, there are security risks associated with this, because the access token may have been stolen after being issued or otherwise alienated from the *Client* to whom it was distributed. As a result, other token types can ask for more guarantees, such as an identity of the *Client* or a client secret. Bearer Token is however the only well standardized and widely used type of token. It does place much responsibility for management of the security risks with *Client* and *Authorisation Server*. This is why Chapter 5 of the specification of the standard RFC6750 explicitly focuses on these security risks and on the measures to deal with them. See for this responsibilities 26, 27 and 28.

The MedMij Framework opts for the method Authorisation Request Header Field because this provides the best security.

8. The *OAuth Client* only uses a single scope at a time. The *OAuth Authorisation Server* generates authorisation codes and access tokens with a single scope that is determined by the *Data Service* to be requested.

Explanatory Notes

The OAuth scope is included in the generation of codes and tokens. This is related to the *Data Service*. Although it is possible in technical terms to include multiple scopes, the scope is limited to a single *Data Service* per request.

9. The *OAuth Authorisation Server* sets the validity duration for each issued authorisation code and each issued access token at precisely 15 minutes (900 seconds). Furthermore, it does not issue any refresh tokens.

Explanatory Notes

This is a measure to counteract the security risks 4.4.1.1 and 4.4.1.3 from RFC 6819 (see under responsibility 26). In addition, the entire flow of *Compile* is executed synchronously (see under

1). The 900 seconds must then be sufficient for the Client to provide the access token to the *Authorisation Server*. A refresh token is then unnecessary.

10a. The *OAuth Authorisation Server* generates authorisation codes and access tokens in such a way that the probability of guessing them is no greater than 2^{-128} with the random number generators used for this being cryptographically secure, too.

10b. If desired, it is permitted to include one or more of the information elements from the following limited list in the authorisation codes and access tokens:

- a progress moment of the validity of the token, under the conditions that both:
    - the value of it is in line with the responsibilities in the MedMij Framework, and
    - its expiry can be used to conclude that the authorisation code or the access token is invalid, this conclusion being drawn by the *Authorisation Server* or the *Resource Server*, but that its validity can **not** be concluded if it has not expired yet, for which a validation of the entire token against the internal records of the *Authorisation Server* is namely the sole authority;
- an identification of the service that issued the token;
- the scope for which the authorisation code or the access token was issued, in the form of a copy of the scope parameter of the authorisation request in response to which the authorisation code or the access token was issued;
- the name of the token's format;
- a digital signature.

10c. No information other than that named in responsibility 10b may be present in the authorisation code or the access token, even if not encrypted. Various options may be taken with respect to the information content of the token between authorisation code and access token. The *OAuth Client* must not interpret the content of the token.

10d. With regard to both authorisation codes and access tokens, the *OAuth Authorisation Server* ensures that two of the same valid ones that it issued are never in circulation.

Explanatory Notes

This is a measure to counteract security risk 4.4.1.3 from RFC 6819 (see under responsibility 26). Two important requirements are to be laid down for the authorisation codes and access tokens that are put into circulation: uniqueness and confidentiality. The requirement of confidentiality weighs heavily in the MedMij Framework. Because the authorisation code (indirectly) and the access token (directly) grant access to personal health information, MedMij opts for a format that is virtually meaningless and that is only given meaning by confrontation with local and properly protected records of the *Authorisation Server*. The maximum permissible probability of guessing it is laid down in RFC6749, section 10.10. It must not be possible to use a comparison of multiple authorisation codes or access tokens to work out how they are generated.

If a progress moment is included in the access token, it becomes possible to get the *Resource Server* to refrain from unnecessary consultation of the Authorisation Server if this is to be implemented separately. The second condition for this option prevents a situation where any corruption in the *Individual's Domain*, namely of the authorisation code or the access token whereby the progress moment would be abandoned, cannot after all lead to wrongful access or to the wrongful placement of health information. The accepting of an authorisation code or an access token always occurs in line with the internal records of the *Authorisation Server*. This corruption can also bring forward the progress moment but this causes little damage. By the way, in the current version of the MedMij Framework, in which the validity duration has a fixed value, the *OAuth Client* will work out itself when it no longer makes sense to still provide an authorisation code or access token. This means that the added value of a progress moment in the authorisation code or the access token can at most be seen in possible future versions.

The service that issued the token is however already a useful information element in this version of the MedMij Framework. In situations where a *Resource Server* works with multiple *Authorisation Servers* that are implemented separately, when provided with an access token it must be able to determine which *Authorisation Server* must be addressed. This addressing can for instance be done by means of Token Introspection in line with RFC7662. The appropriate source for this information is the access token itself, which has information about its origin. This origin information does not cause any additional privacy risks, because the *OAuth Client* knows anyway who it has received the access token from.

Furthermore, the *Authorisation Server* may also include a copy of the scope in (the Authorisation code or) the access token, namely the scope that it previously received in the authorisation request of the *PHE Server* (see Data and performance in UCI Compile and UCI Share, responsibility 5). In this way, the *Resource Server* does not have to be informed separately by the *PHE Server* about the scope. While it's true that the authorisation code or the access token does carry additional meaning, the risks of it are not as great as the risks of letting the *PHE*

Server send the scope separately, which scope could for instance differ from that for which the authorisation code or the access token was issued.

The list of permitted information elements is limited. No other information, not even if encrypted, may be included in the authorisation code or the access token. This exclusion certainly applies to the following too:

- Information about *Individual*;
- Information about *Care Provider or Data Service*, whether or not in relation to *Individual*, outside the scope;
- naming of, and restrictions on, the intended acceptors of the authorisation code or the access token. In respect of this point it is namely the *Care Providers List* that is the authority: if the *OAuth Client* has retrieved an access token at a place that was mentioned in the *Care Providers List* to this end then it must be able to provide this access token at the place that is mentioned in the *Care Providers List* to this end.

The ban on interpretation by the *OAuth Client* of the authorisation code and access token ensures that a minimum dependency is created between the service providers in the individual's domain on the one hand and those in the care provider's domain on the other, so that principles P1 and P7 are complied with as much as possible and internal complexity and implementation choices in the care provider's domain do not filter through to or influence the implementation in the individual's domain.

The limitations of the authorisation code's and the access token's ability to carry a meaning, even if encrypted, promote privacy by means of data minimisation. In addition, they prevent new risks relating to the compromising of this information content. Such compromising would be difficult to discover and ward off in the care provider's domain, if it had been decided to this end to refrain from having internal authorisation records because the information is already being transported in the authorisation code or the access token, via the *OAuth Client*.

11. The *OAuth Client* provides a certain authorisation code a maximum of one time to the *Authorisation Server* to obtain an access token. The *Authorisation Server* removes an authorisation code when it has been offered once for the obtaining of an access token.

Explanatory Notes

This is a measure to counteract security risk 4.4.1 from RFC 6819 (see under responsibility 26). The removal of an authorisation code means that the *Authorisation Server* keeps track of an authorisation code that has been issued once, in order to see whether it has already been used at some time to obtain an access token. If an authorisation code has been provided for a second or subsequent time to obtain an access token then the *Authorisation Server* will reject it and terminate the flow. If the *Client* to whom it was rejected was in bad faith then this has averted a hazard. If however they were in good faith and acted in accordance with the MedMij Framework then they were not the party that had already provided the authorisation code previously, which means there appears to be a security breach.

12. The *OAuth Authorisation Server* only transfers an access token to an *OAuth Client* if the authorisation code provided to this end was issued to this same *OAuth Client*.

Explanatory Notes

This is a measure to counteract security risks 4.4.1.3, 4.4.1.5 and 4.4.1.7 from RFC 6819 (see under responsibility 26). To do this, the *Authorisation Server* must accordingly keep track of which *Clients* it distributes the authorisation codes to. This means that the access token may only be distributed via a redirect URI where the hostname is the same as the hostname of the *OAuth Client* for whom the relevant authorisation code was intended.

13. The *OAuth Client* and *OAuth Authorisation Server* use for their reciprocal transfers PKI (Public Key Infrastructure) certificates, namely server certificates, for the authentication of the other server in an exchange.

Explanatory Notes

This is a measure to counteract security risks 4.4.1.1, 4.4.1.3, 4.4.1.4 and 4.4.1.5 in RFC 6819 (see under responsibility 26). In this release of the MedMij Framework, the PKI certificates are used for two goals in the Network layer: authentication of servers and encryption, which guarantees the confidentiality and integrity of the content of the data transfer.

14. The *OAuth Client* only provides - possibly via de *OAuth User Agent* - to the *OAuth Authorisation Servers* those redirect URIs that are full and that refer to a https-protected endpoint. *OAuth Authorisation Servers* do not redirect to a URI that does not fulfil these requirements.

Explanatory Notes

This is a measure to counteract security risks 4.1.5, 4.2.4, 4.4.1.1, 4.4.1.5 and 4.4.1.6 in RFC 6819 (see under responsibility 26). See in addition the explanatory notes under responsibility 12.

15. The OAuth client type of the *OAuth Client* is confidential.

Explanatory Notes

In order to be able to guarantee privacy, it is important that the OAuth *Authorisation Server* is sufficiently certain about the identity of the OAuth *Client*. This certainty is dependent on the extent to which the OAuth *Client* can keep his credentials confidential. To this end, the OAuth specification distinguishes between two client types: confidential and public. The first type can give the *Authorisation Server* a sufficient degree of confidentiality for his credentials but the second cannot. It is a main aim of MedMij to guarantee such trust in a framework and not to leave this to the individual players. This is why the MedMij Framework links responsibilities to *Clients* so that they can be trusted in respect of *Authorisation Servers*. We expect that a large proportion of the implementations of the Oauth *Client* (i.e. of the *PHE Server*) can provide this confidentiality, because their architecture is what the OAuth specification calls web application. Other types of *PHE Server* architectures, such as those from an app, still continue to be possible but they will be asked to process all data transfers of OAuth client credentials in the background on a server, not on the user device.

Lists

Care Providers List

16. *MedMij Registration* and each *PHE Server* implement the use case *UC Request CPL* with the use case implementation *UCI Request CPL*. They use for this the relevant flowchart.

17. *PHE Server* obtains at least every fifteen minutes (900 seconds) the most recent *CPL implementation* of *MedMij Registration*.

18. *PHE Server* validates each newly obtained *CPL implementation* against the XML schema description of the *Care Providers List*. This XML schema description is a technical implementation of the MedMij meta model.

OAuth Client List

19. *MedMij Registration* and *Authorisation Server* implement the use case *UC Request OCL* with the use case implementation *UCI Request OCL*. They use for this the relevant flowchart.

20. *Authorisation Server* obtains at least every fifteen minutes (900 seconds) the most recent *OCL implementation* of *MedMij Registration*.

21. *Authorisation Server* validates each new *OCL implementation* obtained against the XML schema description of the *OAuth Client List*. This XML schema description is a technical implementation of the MedMij meta model.

Data Service Names List

22. *MedMij Registration*, *PHE Server and Authorisation Server* implement the use case *UC Request DSNL* with the use case implementation *UCI Request DSNL*. They use for this the relevant flowchart.

23. *PHE Server and Authorisation Server* obtain at least every fifteen minutes (900 seconds) the most recent *DSNL implementation* of *MedMij Registration*.

24. *PHE Server and Authorisation Server* validate each new *DSNL implementation* obtained against the XML schema description of the DSNL. This XML schema description is a technical implementation of the MedMij meta model.

Security

25. In the data transfer that takes place in the context of *UCI Compile*, *UCI Share*, *UCI Request CPL*, *UCI Request OCL* and *UCI Request DSNL*, these use the functions *Encryption*, *Server Authentication* and *Server Authorisation*, in line with that laid down in the Network layer.

26. The *OAuth Client* realises the following security measures, in accordance with RFC6819:

security measure	**section in RFC6819**	**mitigated risk(s)**
Clients should use an appropriate protocol, such as OpenID or SAML to implement user login. Both support audience restrictions on clients.	4.4.1.13	4.4.1.13
All clients must indicate their client ids with every request to exchange an authorisation "code" for an access token.		
Keep access tokens in transient memory and limit grants.	5.1.6	
Keep access tokens in private memory.	5.2.2	4.1.3
The "state" parameter should be used to link the authorisation request with the redirect URI used to deliver the access token.	5.3.5	4.4.1.8
CSRF defence and the "state" parameter created with secure random codes should be deployed on the client side. The client should forward the authorisation "code" to the authorisation server only after both the CSRF token and the "state" parameter are validated.		4.4.1.12

27. The *PHE Server* realises the following security measures, in accordance with RFC6819:

security measure	**section in RFC6819**	**mitigated risk(s)**

Client applications should not collect authentication information directly from users and should instead delegate this task to a trusted system component, e.g. the system browser.	4.1.4	4.1.4
The client server may reload the target page of the redirect URI in order to automatically clean up the browser cache.	4.4.1.1	4.4.1.1
If the client authenticates the user, either through a single-sign-on protocol or through local authentication, the client should suspend the access by a user account if the number of invalid authorisation "codes" submitted by this user exceeds a certain threshold.	4.4.1.12	4.4.1.12
Client developers and end users can be educated to not follow untrusted URLs.	4.4.1.8	4.4.1.8
For newer browsers, avoidance of iFrames during authorisation can be enforced on the server side by using the X-FRAME-OPTIONS header. For older browsers, JavaScript frame-busting techniques can be used but may not be effective in all browsers.	5.2.2.6	4.4.1.9
Explain the scope (resources and the permissions) the user is about to grant in an understandable way	5.2.4.2	4.2.2

28. The *OAuth Authorisation Server* realises the following security measures, in accordance with RFC6819:

security measure	**section in RFC6819**	**mitigated risk(s)**
Authorisation servers should consider such attacks: Password Phishing by Counterfeit Authorisation Server	4.2.1	4.2.1
Authorisation servers should attempt to educate users about the risks posed by phishing attacks and should provide mechanisms that make it easy for users to confirm the authenticity of their sites.		
Authorisation servers should decide, based on an analysis of the risk associated with this threat, whether to detect and prevent this threat.	4.4.1.10	4.4.1.10
The authorisation server may force a user interaction based on non-predictable input values as part of the user consent approval.		
The authorisation server could make use of CAPTCHAs.		

The authorisation server should consider limiting the number of access tokens granted per user.	4.4.1.11	4.4.1.11
The authorisation server should send an error response to the client reporting an invalid authorisation "code" and rate-limit or disallow connections from clients whose number of invalid requests exceeds a threshold.	4.4.1.12	4.4.1.12
Given that all clients must indicate their client ids with every request to exchange an authorisation "code" for an access token, the authorisation server must validate whether the particular authorisation "code" has been issued to the particular client.	4.4.1.13	4.4.1.13
Best practices for credential storage protection should be employed.	5.1.4.1	4.4.1.2
Enforce system security measures.	5.1.4.1.1	4.3.2 and 4.4.1.2
Enforce standard SQL injection countermeasures.	5.1.4.1.2	
Store access token hashes only.	5.1.4.1.3	
The authorisation server should enforce a one-time usage restriction.	5.1.5.4	4.4.1.1
If an authorisation server observes multiple attempts to redeem an authorisation "code", the authorisation server may want to revoke all tokens granted based on the authorisation "code".	5.2.1.1	
Bind the authorisation "code" to the redirect URI.	5.2.4.5	4.4.1.3
The authorisation server associates the authorisation "code" with the redirect URI of a particular end-user authorisation and validates this redirect URI with the redirect URI passed to the token's endpoint,		4.4.1.7

Explanatory Notes

When it came to drawing up the responsibilities 26, 27 and 28, use was made of RFC 6819 of IETF, which contains an extensive catalogue of the risks, including a series of measures per risk. Where the risk applies to the usage of OAuth within MedMij, and the measures comply with the MedMij principles, they have been included in the framework.

With regard to the provisions in section 3.1 of RFC 6819 , it can be argued that MedMij proceeds on the following basis:

- handles instead of assertions, so that the *OAuth Resource Server* must be able to refer to data of the *OAuth Authorisation Server;*
- bearer tokens instead of proof tokens. See in this regard responsibility 7 in this layer.

In chapter 4 of RFC 6819 there is an extensive list of security risks. Not applicable are, for the current release of the framework:

- threat 4.1.1: Obtaining Client Secrets, because authentication of OAuth Clients in MedMij works on the basis of PKI server certificates, not on the basis of client secrets;
- threat 4.1.2: Obtaining Refresh Tokens, because the framework does not work with refresh tokens;
- threat 4.2.3: Malicious Client Obtains Existing Authorisation by Fraud, because in the framework there is a strict rule that authorisation (for the time being) may only be used once;
- threat 4.3.4: Obtaining Client Secret from Authorisation Server Database, because authentication of OAuth Clients in MedMij works on the basis of PKI server certificates, not on the basis of client secrets;
- threat 4.3.5: Obtaining Client Secret by Online Guessing, because authentication of OAuth Clients in MedMij is done on the basis of PKI server certificates, not on the basis of client secrets.

The following do indeed apply:

- threat 4.1.3: Obtaining Access Tokens;
- threat 4.1.4: End-user Credential Phished Using Comprised or Embedded Browser;
- threat 4.1.5: Open Redirectors on Client;
- threat 4.2.1: Password Phishing by Counterfeit Authorisation Server;
- threat 4.2.2: User Unintentionally Grants Too Much Access Scope;
- threat 4.2.4: Open Redirector;
- threat 4.3.1: Eavesdropping Access Tokens;
- threat 4.3.2: Obtaining Access Tokens from Authorisation Server Database;
- threat 4.3.3: Disclosure of Client Credentials during Transmission;
- threat 4.4.1.1: Eavesdropping or Leaking Authorisation Code;
- threat 4.4.1.2: Obtaining Authorisation "codes" from Authorisation Server Database;
- threat 4.4.1.3: Online Guessing of Authorisation "codes";
- threat 4.4.1.4: Malicious Client Obtains Authorisation;
- threat 4.4.1.5: Authorisation "code" Phishing;
- threat 4.4.1.6: User Session Impersonation;
- threat 4.4.1.7: Authorisation "code" Leakage through Counterfeit Client;
- threat 4.4.1.8: CSRF against redirect-URI;
- threat 4.4.1.9: Clickjacking Attack against Authorisation;
- threat 4.4.1.10: Resource Owner Impersonation;
- threat 4.4.1.11: DoS Attacks That Exhaust Resources;
- threat 4.4.1.12: DoS Using Manufactured Authorisation "codes";
- threat 4.4.1.13: Code Substitution (OAuth Login).

In relation to the MedMij Framework, the measures that must be taken to mitigate these risks can be broken down into three groups:

- measures which have already been provided for by means of one or more responsibilities in the MedMij Framework. These relate for example to the usage of TLS (Network layer) and DigiD (Application layer) and the limiting of the scope and the validity duration of authorisation codes and access tokens (Application layer);
- measures that despite being suggested by RFC6819 have not been made part of the MedMij Framework, because they do not comply with its principles or with other responsibilities in the system;
- other measures, which still need to be taken by *PHE Server*, *OAuth Client* or *OAuth Authorisation Server*.

The aforementioned responsibilities 26, 27 and 28 mean this last group of measures have become part of the MedMij Framework too.

29. *OAuth Clients*, *Authorisation Server* and *OAuth Resource Server* implement the security measures that apply to these respective roles, in line with section 5.3 of RFC6750.

Explanatory Notes

This responsibility is included because information can be obtained with the bearer token without the identity being checked again. This is why measures must be take to guarantee that the token can only be used correctly.