END-TO-END ENCRYPTION
Known recipient (KeyDepot) - REST
Cookbook
Version 1.2

This document is provided to you free of charge by the

eHealth platform

Willebroekkaai 38 – 1000 Brussel

38, Quai de Willebroeck – 1000 Bruxelles

Table of contents

Table of contents ... 2
1. Document management ... 4
1.1 Document history ... 4
2. Introduction ... 5
2.1 Goal of the service ... 5
2.2 Goal of the document ... 5
2.3 eHealth platform document references ... 5
2.4 External document references ... 5
3. Support ... 6
3.1 For issues in production ... 6
3.2 For issues in acceptance ... 6
3.3 For business issues ... 6
3.4 I.AM Connect ... 6
4. Global overview ... 7
4.1 High-level schemas of the ETEE KeyDepot functionality ... 7
4.1.1 Store a Keypair ... 7
4.1.2 Use a Public Key ... 9
5. Step-by-step ... 11
5.1 Technical requirements ... 11
5.1.1 eHealth platform Authentication ... 11
5.1.2 WS-I Basic Profile 1.1 ... 11
5.1.3 Tracing ... 11
5.2 The KeyDepot Rest Services ... 12
5.2.1 POST /keydepot/attestations/options ... 12
5.2.2 POST /keydepot/attestation/result ... 14
5.2.3 PATCH /keydepot/jwks/{kid} ... 16
5.2.4 GET /keydepot/jwks/ ... 16
5.2.5 GET /keydepot/jwks/{kid} ... 18
5.2.6 DELETE /keydepot/jwks/{kid} ... 19
5.2.7 GET /accounts/{accountId} ... 19
5.2.8 GET/keydepot/keyholder/{kid} ... 20
6. Risks and security ... 22
6.1 Risks & safety ... 22
6.2 Security ... 22
6.2.1 Business security ... 22
6.2.2 The use of username, password and token ... 22
7. Implementation aspects ... 23
7.1 Procedure ... 23
7.1.1 Initiation ... 23
7.1.2 Development and test procedure ... 23

7.1.3 Release procedure 23
7.1.4 Operational follow-up 23
8. Error Management 24
8.1 POST /keydepot/attestations/options 24
8.2 POST /keydepot/attestation/result 24
8.3 PATCH/keydepot/jwks/{kid} 24
8.4 GET /keydepot/jwks/ 24
8.5 GET /keydepot/attestations/{kid} 25
8.6 DELETE /keydepot/jwks/{kid} 25
8.7 GET /accounts/{accountId} 25
8.8 GET/keydepot/keyholder/{kid} 25
8.9 Error codes originating from the eHealth platform: 26
9. Annex 3: Structure of the Attestation Object 27

To the attention of: "IT expert" willing to integrate this web service.

1. Document management

1.1 Document history

Version	Date	Author	Description of changes / remarks
1.0	28/11/2019	eHealth platform	Initial version
1.1	09/04/2020	eHealth platform	WS-I Compliance
1.2	22/04/2021	eHealth platform	Tracing

2. Introduction

2.1 Goal of the service

The End-To-End Encryption (ETEE) basic REST services only offer building blocks that allow integrating secure communications in applications.

It does not offer a pre-packaged 'End-To-End' business solution. This means you have to create your own client application with an implementation of a:

- KeyDepot Client
- software that integrates a cryptographic solution
- way to pass on a message reference to a message receiver
- way to pass on a key reference to a message receiver (optional if a key reference is used in the Message Storage Server (MSS))
- Message Storage Center (you could store the message reference in the MSS);

2.2 Goal of the document

This document is intended as an integration support guide for the eHealth platform's REST service "ETEE - Depot Interface". The target audience is software integrators implementing the ETEE REST service in their own custom application. This document is not a software manual for end users but explains the concepts, principles and interface of the KeyDepot REST WS.

2.3 eHealth platform document references

On the portal of the eHealth platform, you can find all the referenced documents.[1]. These versions or any following versions can be used for the eHealth platform service.

ID	Title	Version	Date	Author
1	Glossary.pdf	1.0	Pm	eHealth platform
2	eHealth Services – Web Access	2.0	12/07/2019	eHealth platform
3	I.AM Connect Technical specifications	1.1	12/08/2019	eHealth platform
4	I.AM Connect – Client Registration	1.02	25/02/2019	eHealth platform

2.4 External document references

ID	Title	Source
1	Webauthn site	https://www.w3.org/TR/webauthn/
2	Basic Profile Version 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[1] ***www.ehealth.fgov.be/ehealthplatform***

3. Support

3.1 For issues in production

eHealth platform contact center:

- Phone: 02/788 51 55
- Mail: ***support@ehealth.fgov.be***
- *Contact Form :*
  - ***https://www.ehealth.fgov.be/ehealthplatform/nl/contact*** (Dutch)
  - ***https://www.ehealth.fgov.be/ehealthplatform/fr/contact*** (French)

3.2 For issues in acceptance

Integration-support@ehealth.fgov.be

3.3 For business issues

- regarding an existing project: the project manager in charge of the application or service
- regarding a new project and other business issues: ***info@ehealth.fgov.be***

3.4 I.AM Connect

- In order to use the KeyDepot REST service you have to obtain an "Access token" which is delivered through I.AM Connect. You can find more information about I.AM Connect and how to register a client in I.AM Connect on the IAM eHealth portal page:

  Dutch version: ***https://www.ehealth.fgov.be/ehealthplatform/nl/service-iam-identity-access-management***

  French version: ***https://www.ehealth.fgov.be/ehealthplatform/fr/service-iam-identity-access-management***

4. Global overview

4.1 High-level schemas of the ETEE KeyDepot functionality

Below you will find two schemas representing the context of the 'Depot Interface' REST service use. First to store and secondly an example to get a recording.

4.1.1 Store a Keypair

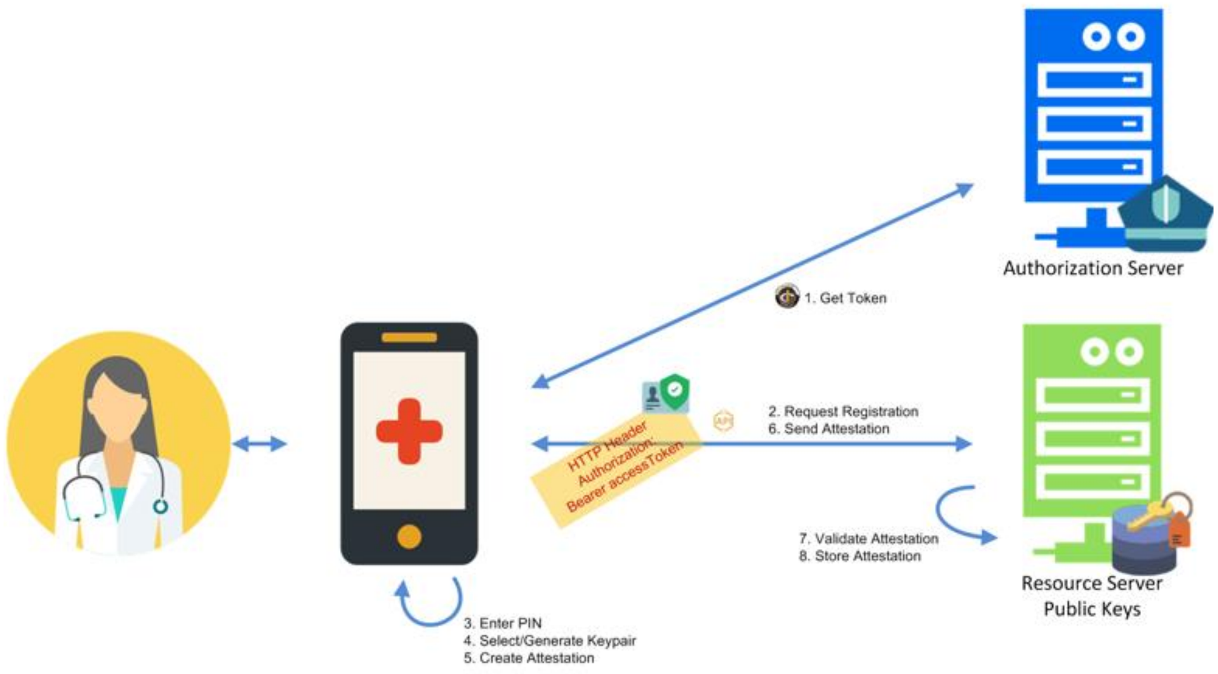


Figure 4. Public Key Registration

1. The client/app requests authorization at the eHealth platform. See authentication/authorization flows for a full description.
2. If the client has received permissions to register asymmetric keys on behalf user, this will be mentioned in the access Token and the client/app can send a request for a registration of a new public key (initial registration request).

   The eHealth platform follows the W3C Web Authentication recommendation (W3C WebAuthn) to register keys. The initial request for registration is outside of the scope of WebAuthn. This means the eHealth platform will return a challenge, user info, and relying party info to the client. W3C WebAuthn.
3. Before doing anything, the client or device that will generate the key pair will ask for some form of user verification, like a PIN, to prove that the user is present and consenting the registration.
4. After the user verification, the client or device will create a new asymmetric key pair and safely store the private key for future reference.
5. The client or device will generate and sign an attestation, including the newly generated public key.

   The attestation can be signed by the generated key itself (self-attestation) or the device that generated the key in case it supports this (such as a WebAuthn compliant Authenticator). The latter can be used to prove that the key pair was generated and protected by a device, recognized to be secure.

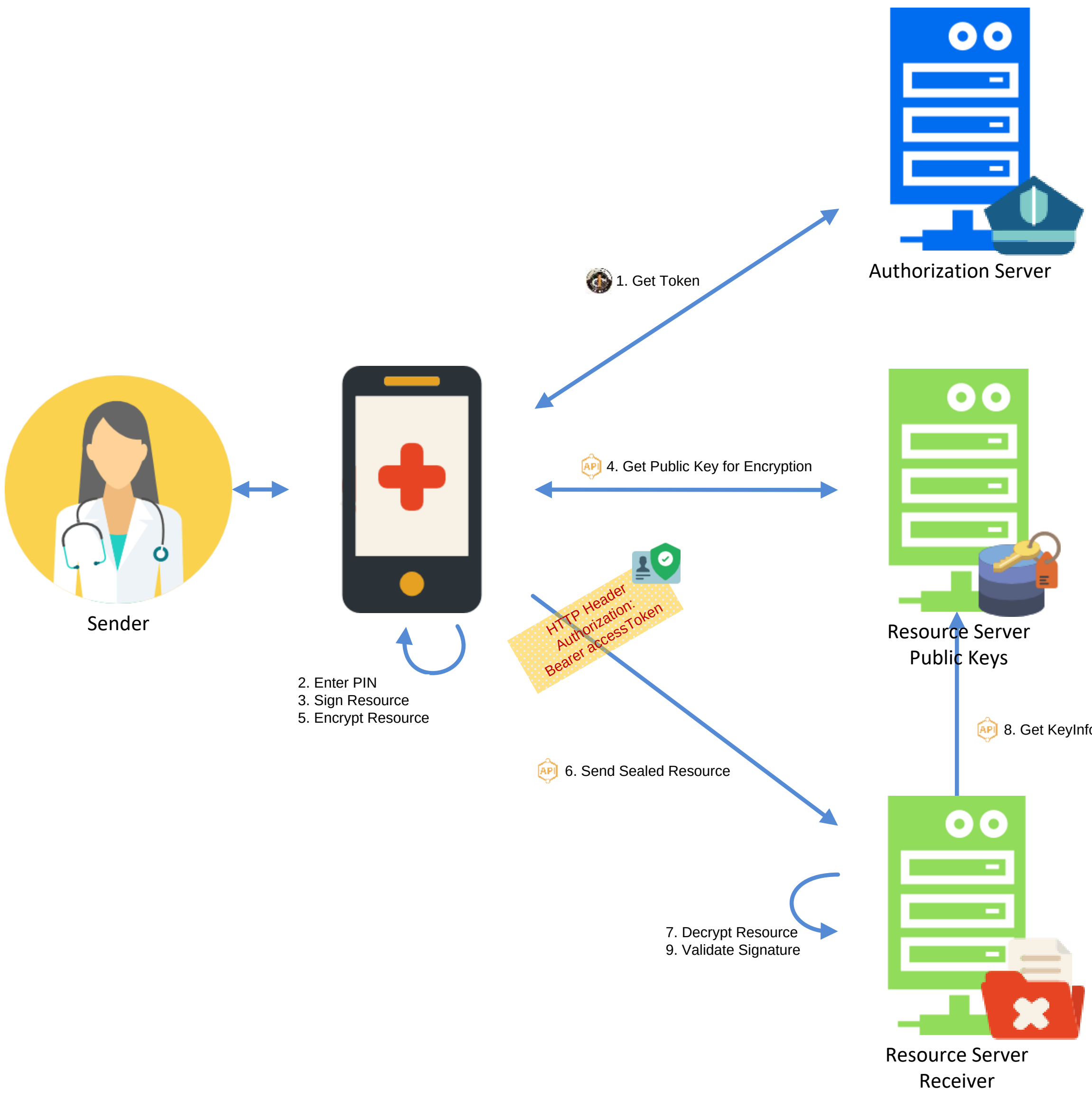
The client should also offer the user to choose a meaningful name for his key for later consultation and revocation. This name will be transmittable to the eHealth key store.

6. The client sends the attestation to the eHealth platform.
7. The eHealth platform will validate the attestation to ensure that the registration was complete and not tampered with.

   Validation includes the challenge, origin, and signature. If a recognized authenticator device signed the attestation, that chain will be validated as well.
8. Assuming that the checks pan out, the eHealth platform will store the new public key associated with the user's account for future use.

   The key will be linked to the user and the requesting client, both mentioned in the access Token: Identifier (Id, Type), application Identifier. eHealth's Keystore API will have search options for those fields, including the unique credentialId of the key, generated by the client or authenticator device, as described in the WebAuthn recommendation. An expiration policy is added to the key.

4.1.2 Use a Public Key



1. The client requests authorization at the eHealth platform. See authentication/authorization flows for a full description.
2. The access Token returned to the client will contain a reference to the registered, active key(s) linked to the combination of the given client-user, for future reference. See section 'Security Recommendations, Risks & Known Limitations' *in document 2.3 eHealth Services – Web Access.*
3. The client requests the user to unlock his private key.
4. The client signs the message for content integrity and message authentication.

   The signature must be placed with a key of the user, registered at the eHealth platform and active for the given client. Those are referenced in the access Token, received in step 1. This ties the pieces together: key, client and end-user.
5. The client gets the public key of the receiver that can be used for encrypting messages to the particular resource.

   How the public key is selected from the resource server depends on how the project/app/client is setup to send messages from sender to receiver. The Public Keys Resource Server will support sufficient filtering *options, such as: unique identifier of the receiver (SSIN), application identifier (clientID), usage (enc). The user should only be asked to select a key if there are multiple receivers to choose from.*

6. The client encrypts the message with the key received in step 4
7. The client sends the sealed message to the receiver. The access Token, received at the end of step 1, is added to the request.
8. If the access Token contains sufficient privileges, the receiver decrypts the message.
9. To validate the signature on the message and to authenticate the owner of the key used for it, the receiver can use the reference of the key to get it from eHealth's key store, verify the signature with it and verify if the reference is claimed in the access Token, which proves it is a valid one, linked to the claimed user. This verifies that the sender of the message is also the author and that the receiver was the intended audience of the original author.
10. The receiver validates the signature.

5. Step-by-step

5.1 Technical requirements

5.1.1 eHealth platform Authentication

As explained previously, to use the ETEE KeyDepot service, you must have an access token delivered through I.AM Connect.

One role is defined for the using of the KeyDepot Rest service:

- manage-keys: This role must be present in the access token in order to use the POST, PATCH and DELETE methods of the service

<u>Presentation of the roles in the access token:</u>

```
"ehealth-etee-backend": {
  "roles": [
    "read-keys",
    "manage-keys"
  ]
}
```

For the organizations using the KeyDepot Rest service, the local manager of the organization has the possibility to assign these roles to his members through the UMAN application.

5.1.2 WS-I Basic Profile 1.1

Your request must be WS-I compliant (Cfr External Ref). If not you will receive one of the errors SOA-03001 – SOA-03003.

5.1.3 Tracing

To use this service, the request SHOULD contain the following two http header values (see RFC ***https://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.8***):

1. **User-Agent**: information identifying the software product and underlying technical stack/platform.

   - Pattern: {company}/{package-name}/{version} {platform-company}/{platform-package-name}/{platform-package-version}
   - Regular expression for each subset (separated by a space) of the pattern: [[a-zA-Z0-9-\/]*\/[0-9a-zA-Z-_.]*
   - Examples:
     User-Agent: MyCompany/myProduct/62.310.4 eHealth/Technical/3.19.0
     User-Agent: Topaz-XXXX/123.23.X Taktik/freeconnector/XXXXX.XXX

2. **From:** email-address that can be used for emergency contact in case of an operational problem
Examples:
From: info@mycompany.be

5.2 The KeyDepot Rest Services

The REST interface is described with a JSON/Swagger API.

To store a new Keypair, you need first to send a POST-options request and then a POST-result request.

5.2.1 POST /keydepot/attestations/options

- ***Request***

Requests "authorization" options.

This initial request is necessary to retry information (be authenticated) before send an attestationObject or JsonObject to register a key.

If the client has received permissions to register asymmetric keys on behalf user, the request will return a success response.

Request elements:

Element	Description
username	String of the username
displayName	String of the display name
authenticatorSelection	
attestation	

Example:

```
{
  "username": "apowers",
  "displayName": "Adam Powers"
  "authenticatorSelection":
  {
  }
}
```

- ***Response***

The POST operation returns an attestation authorisation, containing a challenge, user info and relying party info to the client. **200 ok**

Element	Description
attestation	Specify their preference regarding attestation conveyance during credential generation: - "none": Relying Party is not interested in authenticator attestation - "indirect": Relying Party prefers an attestation conveyance yielding verifiable attestation statements, but allows the client to decide how to obtain such attestation statements - "direct": Relying Party wants to receive the attestation statement as generated by the authenticator

challenge	This member represents a challenge that the selected authenticator signs, along with other data, when producing an authentication assertion
authenticatorSelection	<u>authenticatorAttachment:</u> If this member is present, eligible authenticators are filtered to only authenticators attached with the specified value (“platform” or “cross-platform” or unspecified) <u>userVerification:</u> This member describes the Relying Party's requirements regarding user verification for the create() operation. Eligible authenticators are filtered to only those capable of satisfying this requirement. (Available values: “required”, “preferred”, “discouraged”) <u>requiredResidentKey:</u> This member describes the Relying Parties requirements regarding resident credentials. If the parameter is set to true, the authenticator MUST create a client-side-resident public key credential source when creating a public key credential.
rp	This member contains data about the Relying Party responsible for the request
timeout	This member specifies a time, in milliseconds, that the caller is willing to wait for the call to complete
publicKeyCredParams	Information about the desired properties of the credential to be created - Type: the type of credential to be created - Alg: the cryptographic signature algorithm with which the newly generated credential will be used, and thus also the type of asymmetric key pair to be generated, e.g., RSA or Elliptic Curve
user	Information about the user who sent the request: - username: the username used in the request - displayname: the displayname used in the request - accountId: Id generated by the service and linked to the user

<u>Example</u>:

```
{
  "rp": {
    "id": null,
    "name": "ci-etee-webapp-client",
    "icon": null
  },
  "challenge": "ZjBhYjJmYjktM2RhZC00NTg3LWEwZjktZDYwNzVmYzNmZmMw",
  "pubKeyCredParams": [
    {
      "type": "public-key",
      "alg": -7
    },
    {
      "type": "public-key",
      "alg": -257
    }
  ],
  "timeout": 300000,
  "authenticatorSelection": null,
  "attestation": "direct",
  "user": {
    "name": "name",
    "displayName": "displayName",
    "id": "MDY2MmIzZDktMjIyMy00ZTI5LThlNTYtY2Y0YmMwMDk0YzYx"
  }
}
```

5.2.2 POST /keydepot/attestation/result

- ***Request***

Post request to send an attestation to eHealth. This attestation contains mainly the public key of an 'assymetric keys pair' as object.
/!\ This request contains only simple attestation. To extend the key with more information, it is necessary to use de request PATCH /keydepot/jwks/{kid}

This request has inputs:

Element	**Description**
id	The credential's identifier. the base64url encoding of the rawid
rawId	This attribute returns the ArrayBuffer contained in the [[identifier]] internal slot. This internal slot contains the credential ID, chosen by the authenticator. The credential ID is used to look up credentials for use, and is therefore expected to be globally unique with high probability across all credentials of the same type, across all authenticators.
type	The PublicKeyCredential interface object's [[type]] internal slot's value is the string "public-key".
clientDataJSON	This attribute contains a JSON serialization (This is the result of JSON stringifying and UTF-8 encoding to bytes a CollectedClientData dictionary) of the client data passed to the authenticator by the client in its call to either create() or get()
attestationObject	This attribute contains an attestation object, which is opaque to, and cryptographically protected against tampering by, the client. The attestation object

	contains both authenticator data and an attestation statement. The former contains the AAGUID, a unique credential ID, and the credential public key. The contents of the attestation statement are determined by the attestation statement format used by the authenticator. It also contains any additional information that the Relying Party's server requires to validate the attestation statement, as well as to decode and validate the authenticator data along with the JSON-serialized client data

Example:

```
{
	"id": "dmggefOw7Zdso8sGTR_VW42YtBIaxEYVJE7PMs3p62ysGRRJJzWl0fPNkI6bXUT-	"rawid": "dmggefOw7Zdso8sGTR_VW42YtBIaxEYVJE7PMs3p62ysGRRJJzWl0fPNkI6bXUT-QQP35uXhB9bDxCC-lZTIww",
	"type": "public-key",
	"clientDataJSON": "eyJjaGFsbGVuZ2UiOiJaYTFaRzYxREZhc3pQeW00Tng5TG1iUUVHZkZ1RXVQNWpxRHgtckFRd191RGtRd
XYyZDVsVUcyVmZlS3lOMGJkc0kzRUJUeG02d0lKamxSdDFWVURhZyIsIm5ld19rZXlzX21heV9iZV9hZGRl
ZF9oZXJlIjoiZG8gbm90IGNvbXBhcmUgY2xpZW50RGF0YUpTT04gYWdhaW5zdCBhIHRlbXBsYXRlLiBTZW
UgaHR0cHM6Ly9nb28uZ2wveWFiUGV4Iiwib3JpZ2luIjoiaHR0cHM6Ly93ZWJhdXRobi5vcmciLCJ0eXBlIjoid
2ViYXV0aG4uY3JlYXRlIn0",
	 "attestationObject": "o2NmbXRmcGFja2VkZ2F0dFN0bXSjY2FsZyZjc2lnWEgwRgIhALiD9zcYn70iduuqf5vIs-GQUBsKOgYuIpWi-
N7uZ3J2AiEAnOw7OdjvufPpI5B90voC_9rHEL6lXZyvgNWZHTVbHD1jeDVjgVkCwjCCAr4wggGmoAMCAQ
ICBHSG_cIwDQYJKoZIhvcNAQELBQAwLjEsMCoGA1UEAxMjWXViaWNvIFUyRiBSb290IENBIFNlcmlhbCA0
NTcyMDA2MzEwIBcNMTQwODAxMDAwMDAwWhgPMjA1MDA5MDQwMDAwMDBaMG8xCzAJBgNVBA
YTAlNFMRIwEAYDVQQKDAlZdWJpY28gQUIxIjAgBgNVBAsMGUF1dGhlbnRpY2F0b3IgQXR0ZXN0YXRpb2
4xKDAmBgNVBAMMH1l1YmljbyBVMkYgRUUgU2VyaWFsIDE5NTUwMDM4NDIwWTATBgcqhkjOPQIBBg
gqhkjOPQMBBwNCAASVXfOt9yR9MXXv_ZzE8xpOh4664YEJVmFQ-
ziLLl9lJ79XQJqlgaUNCsUvGERcChNUihNTyKTlmnBOUjvATevto2wwajAiBgkrBgEEAYLECgIEFTEuMy42LjEu
NC4xLjQxNDgyLjEuMTATBgsrBgEEAYLlHAIBAQQEAwIFIDAhBgsrBgEEAYLlHAEBBAQSBBD4oBHzjApNFYA
GFxEfntx9MAwGA1UdEwEB_wQCMAAwDQYJKoZIhvcNAQELBQADggEBADFcSIDmmlJ-
OGaJvWn9CqhvSeueToVFQVVvqtALOgCKHdwB-Wx29mg2GpHiMsgQp5xjB0ybbnpG6x212FxESJ-
GinZD0ipchi7APwPlhIvjgH16zVX44a4e4hOsc6tLIOP71SaMsHuHgCcdH0vg5d2sc006WJe9TXO6fzV-
ogjJnYpNKQLmCXoAXE3JBNwKGBIOCvfQDPyWmiiG5bGxYfPty8Z3pnjX-
1MDnM2hhr40ulMxlSNDnX_ZSnDyMGIbk8TOQmjTF02UO8auP8k3wt5D1rROIRU9-
FCSX5WQYi68RuDrGMZB8P5-
byoJqbKQdxn2LmE1oZAyohPAmLcoPO5oYXV0aERhdGFYxJVpCI8ezuMjKVQDXb0Q18rjkTBaJ1G1WbuP1
8uyKb3UQQAAAAP4oBHzjApNFYAGFxEfntx9AEB2aCB587Dtl2yjywZNH9VbjZi0EhrERhUkTs8yzenrbKwZF
EknNaXR882QjptdRP5BA_fm5eEH1sPEIL6VlMjDpQECAyYgASFYIOvQEknPbz1SUGSYult0AZu3tMbTZALIf-
Ag28do-hHsIlggzNfCQUrlO8MZ2umPFs_tAALs0yczVeLc7fwuEwNMF8g"
}
```

- ***Response***

The response contains the status of the creation: by example if success: **201 created**

5.2.3 PATCH /keydepot/jwks/{kid}

PATCH method to add information to an existing public key already registered by a POST method.

- ***Request***

The method specifies **Kid** to recognize the key that we want to patch.

All the inputs of the request are optional:

- An **'use'** specification if the public key has a limitation to a specific usage.
- A meaningfull **name** for this key.

Example:
```
{
  "use": "sig",
  "name": "moto g 2013"
}
```

Response

Just a **200 success** or an error.

5.2.4 GET /keydepot/jwks/

GET method to retrieve all the user's public keys for a specified usage.

- ***Request***

Element	Description
type	Id type of the client (SSIN, NIHII, CBE, EHP)
identifier	Value of the client's identifier
use	The use specification of the public key(s) which must be retrieved: 2 values possible: "sig" or "enc"
application	The application name of the public key(s) which must be retrieved
validityTime	This parameter is a date. It allows filtering the response in order to return only the key(s), which was (were) valid at this time.

Example:

GET https://testURL/etee/v1/pubKeys/webauthn/jwks?type=NIHII&value=12345678&use=enc

- ***Response***

A **200 success** or an error + the element key composed by the following elements:

Element	Description
kty	The key type
crv	The curve of the key
use	The use specification of the public key which must was retrieved: 2 values possible: "sig" or "enc"

alg	The signature algorithm
kid	The identifier of the key (its credential ID)

Example:

```
{
  "keys": [
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
      "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
      "use": "enc",
      "kid": 1
    },
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
      "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
      "use": "enc",
      "kid": 1
    }
  ]
}
```

GET /keydepot/attestations/{kid}

GET method to retrieve the attestationObject linked to the public key

- ***Request***

Element	Description
kid	The identifier of a key (its credential ID)

- ***Response***

Element	Description
attestationObject	This attribute contains an attestation object, which is opaque to, and cryptographically protected against tampering by, the client. The attestation object contains both authenticator data and an attestation statement. The former contains the AAGUID, a unique credential ID, and the credential public key. The contents of the attestation statement are determined by the attestation statement format used by the authenticator. It also contains any additional information that the Relying Party's server requires to validate the attestation statement, as well as to decode and validate the authenticator data along with the JSON-serialized client data

Example:

```
{
  "fmt": "packed",
  "attStmt":   {
    "alg": -7,
    "sig":
"MEUCIChhscYVuwVJ7auffigqP0aQAAka5xdLWNqxPXohL+AJAiEAtq4O03hodwiA6UwXJpinut/IumEXiJZbh6XC6
Mx5Km8=",
    "x5c":
["MIICvDCCAaSgAwIBAgIEA63wEjANBgkqhkiG9w0BAQsFADAuMSwwKgYDVQQDEyNZdWJpY28gVTJGIFJvb3Qg
Q0EgU2VyaWFsIDQ1NzIwMDYzMTAgFw0xNDA4MDEwMDAwMDBaGA8yMDUwMDkwNDAwMDAwMFowbTEL
MAkGA1UEBhMCU0UxEjAQBgNVBAoMCVl1YmljbyBBQjEiMCAGA1UECwwZQXV0aGVudGljYXRvciBBdHRlc3Rhd
GlvbjEmMCQGA1UEAwwdWXViaWNvIFUyRiBFRSBTZXJpYWwgNjE3MzA4MzQwWTATBgcqhkjOPQIBBggqhkjOP
QMBBwNCAAQZnoecFi233DnuSkKgRhalswn+ygkvdr4JSPltbpXK5MxlzVSgWc+9x8mzGysdbBhEecLAYfQYqpVLW
WosHPoXo2wwajAiBgkrBgEEAYLECgIEFTEuMy42LjEuNC4xLjQxNDgyLjEuNzATBgsrBgEEAYLlHAIBAQQEAwIEMDA
hBgsrBgEEAYLlHAEBBAQSBBD6K5ncnjlCV4+SSjDSPEEYMAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQELBQADg
gEBACjrs2f+0djw4onryp/22AdXxg6a5XyxcoybHDjKu72E2SN9qDGsIZSfDy38DDFr/bF1s25joiu7WA6tylKA0HmEDl
oeJXJiWjv7h2Az2/siqWnJOLic4XE1lAChJS2XAqkSk9VFGelg3SLOiifrBet+ebdQwAL+2QFrcR7JrXRQG9kUy76O2VcS
gbdPROsHfOYeywarhalyVSZ+6OOYK/Q/DLIaOC0jXrnkzm2ymMQFQlBAIysrYeEM1wxiFbwDt+lAcbcOEtHEf5ZlWi
75nUzlWn8bSx/5FO4TbZ5hIEcUiGRpiIBEMRZlOIm4ZIbZycn/vJOFRTVps0V0S4ygtDc="]
  },
  "authData":
"qMoabMVg0shaCPj1W89eumXXnbFImqNaojrhUE/2VkpBAAAARPormdyeOUJXj5JKMNI8QRgAQDT9JVDZAEEjE
VRse8BsAIL0l8zGC3NQ3MQ3YgteevbTF+6F4GcFy+1izsB7/arHgy6dDCrW75QWy5PV83sFFmmlAQIDJiABIVggu9H
JpHA0+HqQKCk0e/h/QVs36aKl44FzTw5y1ww2ARciWCDUt9wTtRg8yZuslA66rIQ2A71WjoxFWKefx3wlqJeA/A=="
}
```

5.2.5 GET /keydepot/jwks/{kid}

GET method to retrieve only one specific public key.

- ***Request***

Element	Description
kid	The identifier of a key (its credential ID)

- ***Response***

A **200 success** or an error + the element key which is composed by the following elements:

Element	Description
kty	The key type
crv	The curve of the key
use	The use specification of the public key which must was retrieved: 2 values possible: “sig” or “enc”
kid	The identifier of the key (its credential ID)

Example:
```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
  "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
  "use": "enc",
  "kid": 1
}
```

5.2.6 DELETE /keydepot/jwks/{kid}

DELETE method to delete a specific public key.

- ***Request***

Element	Description
kid	The identifier of a key (its credential ID)

- ***Response***

Just a **200 success** response or an error.

5.2.7 GET /accounts/{accountId}

GET method to retrieve all the keys linked to a person (public keys, keys created,…)

- ***Request***

Element	Description
accountId	Account Id linked to the user. It was generated and returned to the user in the POST/keydepot/attestations/options response

- ***Response***

Element	Description
accountid	Account Id that the user inserted in his request
username	The username linked to the user
details - name	Meaningfull name of the key
details - attestationObjectRef	Reference to the AttestationObject of the public keys linked to the accountId
details - jwkRef	References to the public keys linked to the accountId

Example:
```
{
  "accountId": "08455776-6b71-45a9-bf07-9f98d3776dc3",
  "details": [
    {
      "jwkRef": "{}",
      "name": "moto g 2013",
      "attestationObjectRef": "{}"
    },
    {
      "jwkRef": "{}",
      "name": "moto g 2013",
      "attestationObjectRef": "{}"
    }
  ],
  "username": "johndoe"
}
```

5.2.8 GET/keydepot/keyholder/{kid}

Get method which allows to retrieve the information about a key holder

- ***Request***

Element	Description
kid	The identifier of a key (its credential ID)

- ***Response***

Element	Description
keyholder	Claim of the user

Example:

```
{
  "keyholder": [
    {
      "persons": [{
            "ssin": "0123456789",
            "lastName": "Doe",
            "firstName": "John",

      }]
}

  ]
}
```

OR

```
{
  "keyholder": [
    {
      "organizations": [{
                  "name": "PHARMACY TOUJOURS MALADE",
                  "pharmacy": {
                        "nihii": "12345678",

                  }
            }] }

  ]
}
```

6. Risks and security

6.1 Risks & safety

6.2 Security

6.2.1 Business security

In case the development adds an additional use case based on an existing integration, the eHealth platform must be informed at least one month in advance with a detailed estimate of the expected load. This will ensure an effective capacity management.

In case of technical issues on the WS, the partner may obtain support from the contact center (see Chap 3)

In case the eHealth platform finds a bug or vulnerability in its software, we advise the partner to update his application with the newest version of the software within 10 business days.

In case the partner finds a bug or vulnerability in the software or web service that the eHealth platform delivered, he is obliged to contact and inform us immediately. He is not allowed to publish this bug or vulnerability in any case.

6.2.2 The use of username, password and token

The username, pass word and token are strictly personal. Partners and clients are not allowed to transfer them. Every user takes care of his username, pass word and token and he is forced to confidentiality of it. Moreover, every user is responsible for every use, which includes the use by a third party, until the inactivation.

7. Implementation aspects

7.1 Procedure

This chapter explains the procedures for testing and releasing an application in acceptation or production.

7.1.1 Initiation

If you intend to use the eHealth platform service, please contact ***info@ehealth.fgov.be***. The project department will provide you with the necessary information and mandatory documents.

7.1.2 Development and test procedure

You have to develop a client in order to connect to our WS. Most of the required integration info to integrate is published on the portal of the eHealth platform.

Upon request, the eHealth platform provides you in some cases, with a mock-up service or test cases in order for you to test your client before releasing it in the acceptance environment.

7.1.3 Release procedure

When development tests are successful, you can request to access the acceptance environment of the eHealth platform. From this moment, you start the integration and acceptance tests. The eHealth platform suggests testing during minimum one month.

After successful acceptance tests, the partner sends his test results and performance results with a sample of “eHealth request” and “eHealth answer” by email to his point of contact at the eHealth platform.

Then the eHealth platform and the partner agree on a release date. The eHealth platform prepares the connection to the production environment and provides the partner with the necessary information. During the release day, the partner provides the eHealth platform with feedback on the test and performance tests.

For further information and instructions, please contact: ***integration-support@ehealth.fgov.be***.

7.1.4 Operational follow-up

Once in production, the partner using the eHealth platform service for one of his applications will always test first in the acceptance environment before releasing any adaptations of its application in production. In addition, he will inform the eHealth platform on the progress and test period.

8. Error Management

8.1 POST /keydepot/attestations/options

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	Error Code	Error description
400 (Bad Request)	BAD_REQUEST	The request is not valid
401	NOT_AUTHENTICATED	The JsonWebToken is not valid.
503	Service Unavalaible	Failure server response

8.2 POST /keydepot/attestation/result

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	Error Code	Error description
400 (Bad Request)	BAD_REQUEST	The request is not valid
401	NOT_AUTHENTICATED	Client error
412	VALIDATION_FAILED	Error in challenge, origin and signature validation
503	Service Unavalaible	Failure server response

The Swagger schema validation of the request and response objects will be done by the API Gateway.

A validation is made on attestation sent by post request.

8.3 PATCH/keydepot/jwks/{kid}

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	Error Code	Error description
400 (Bad Request)	BAD_REQUEST	The request is not valid
404	NOT FOUND	A Specific resource was requested and not found
503	Service Unavalaible	Failure server response

8.4 GET /keydepot/jwks/

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	Error Code	Error description
400 (Bad Request)	BAD_REQUEST	The request is not valid

404	NOT FOUND	A Specific resource was requested and not found
200	Empty list	The search request returns no result

8.5 GET /keydepot/attestations/{kid}

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	**Error Code**	**Error description**
400 (Bad Request)	BAD_REQUEST	The request is not valid
404	NOT FOUND	A Specific resource was requested and not found
204	No Content	The search request returns no result

8.6 DELETE /keydepot/jwks/{kid}

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	**Error Code**	**Error description**
400 (Bad Request)	BAD_REQUEST	The request is not valid
404	NOT FOUND	A Specific resource was requested and not found
204	No Content	The search request returns no result

8.7 GET /accounts/{accountId}

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

Http error code	**Error Code**	**Error description**
400 (Bad Request)	BAD_REQUEST	The request is not valid
404	NOT FOUND	A Specific resource was requested and not found
204	No Content	The search request returns no result

8.8 GET/keydepot/keyholder/{kid}

The request body should be valid as defined in the REST interface (Swagger). If not, the WS must return an error 400 : Bad request.

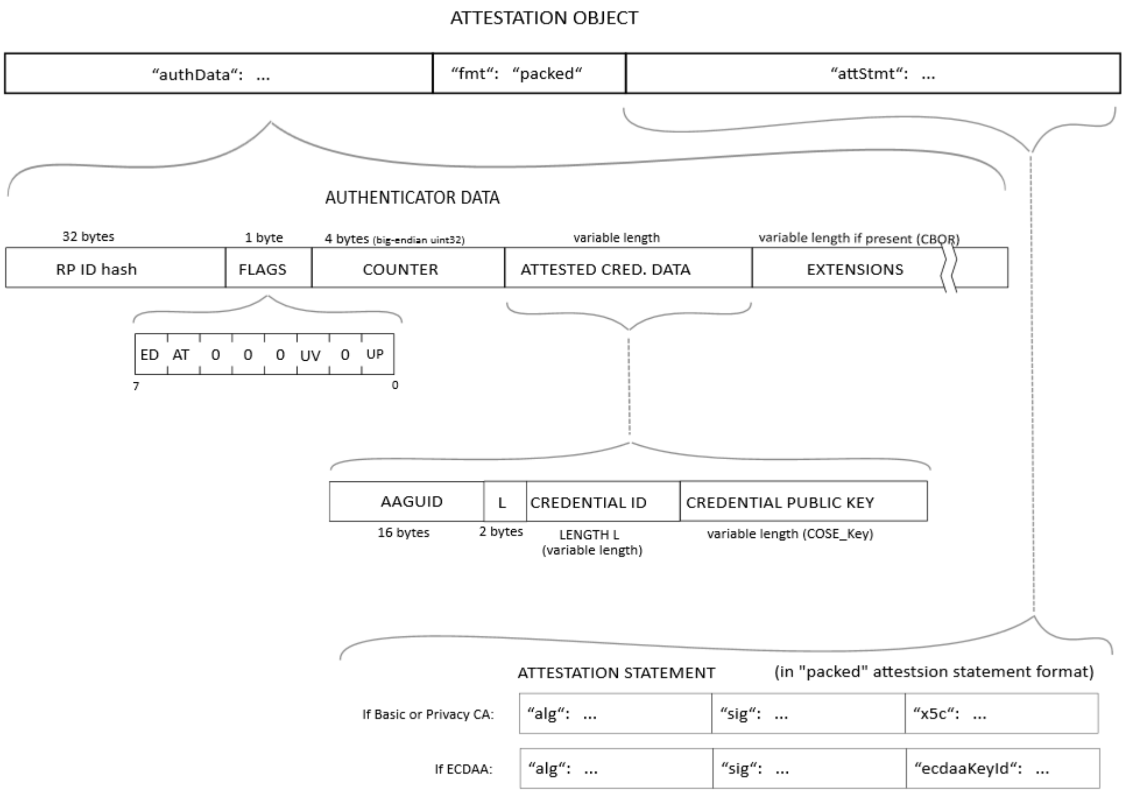
Http error code	Error Code	Error description
400 (Bad Request)	BAD_REQUEST	The request is not valid
404	NOT FOUND	A Specific resource was requested and not found
204	No Content	The search request returns no result

8.9 Error codes originating from the eHealth platform:

These error codes first indicate a problem in the arguments sent, or a technical error.

Error code	Component	Description	Solution
SOA-03001	*Consumer*	*This is the default error for content related errors in case no more details are known.*	***Malformed message***
SOA-03002	*Consumer*	*Message does not respect the SOAP standard.*	***Message must be SOAP***
SOA-03003	*Consumer*	*Message respects the SOAP standard, but body is missing.*	***Message must contain SOAP body***

9. Annex 3: Structure of the Attestation Object



ATTESTATION OBJECT
"authData": ...
"fmt": "packed"
"attStmt": ...
AUTHENTICATOR DATA
32 bytes
1 byte
4 bytes (big-endian uint32)
variable length
variable length if present (CBOR)
RP ID hash
FLAGS
COUNTER
ATTESTED CRED. DATA
EXTENSIONS
ED AT 0 0 0 UV 0 UP
7
0
AAGUID
L
CREDENTIAL ID
CREDENTIAL PUBLIC KEY
16 bytes
2 bytes
LENGTH L (variable length)
variable length (COSE_Key)
ATTESTATION STATEMENT
(in "packed" attestsion statement format)
If Basic or Privacy CA:
"alg": ...
"sig": ...
"x5c": ...
If ECDAA:
"alg": ...
"sig": ...
"ecdaaKeyId": ...