Service public fédéral Intérieur
Direction générale Institutions et Population
Registre national
Gestion des Clients

Au Collège communal / Collège des Bourgmestre et Echevins
A l'attention du service population

Aux sociétés informatiques

Aux fournisseurs de services informatiques

Votre correspondant	**T**	**Votre référence**	**Annexes**
Jean Cuvelier	02 518 22 90		3
E-mail	**F**	**Notre référence**	**Bruxelles**
Jean.cuvelier@rrn.fgov.be	02 518 27 90	III	4 novembre 2020

Registre national des personnes physiques. – Basculement vers la 3ième génération des webservices.

Mesdames,
Messieurs,

Par le biais de la présente note, nous souhaitons vous informer des développements et de l'évolution dans le domaine des webservices du Registre national.

1. Contexte.

La modernisation des services du Registre national est une préoccupation constante afin d'assurer une communication avec nos utilisateurs web, d'une qualité et d'une sécurité optimales.

La deuxième génération des webservices est apparue en 2012 et marquait la fin définitive du protocole X25 utilisé depuis des dizaines d'années. A l'époque, elle a également permis de mettre en œuvre de nouvelles politiques d'accès (NN, CN, EN), autorisant l'authentification à l'aide du numéro d'entreprise (dans le cadre des cercles de confiance) en plus de l'authentification par le numéro national.

En raison de l'évolution de la technologie et de la sécurité, cette 2ième génération peut à présent laisser la place à la 3ième génération qui est dès à présent disponible et opérationnelle pour l'utilisation de tous les services du Registre national. Les deux générations cohabitent déjà en ce moment.

Dans un avenir plus lointain et en vue de préparer l'évolution vers la quatrième génération des webservices du Registre national, la 2ième génération disparaîtra le 1er juillet 2023, seule la troisième génération sera maintenue à ce moment en service.

Ensuite, le remplacement de la 3ième génération sera envisagé aux alentours de 2028.

2. Caractéristiques de la 3ième génération des webservices.

- Par rapport à la 2ième génération , toutes les fonctionnalités sont conservées.

- Améliorations par rapport à la précédente génération :

  - Au niveau des normes et de l'interopérabilité : le protocole services web SOAP est remplacé par le protocole services web REST (ou RESTful). Ce dernier protocole est plus récent et plus flexible.

  - Au niveau sécurité, les webservices acceptent maintenant des certificats de type EC (elliptic curve) en plus des certificats type RSA. Le niveau de sécurité est plus élevé.

  - Au niveau du suivi (tracing) de la réponse à une requête, la présence d'un identifiant spécifique dans la requête est, à présent, possible. Ce dernier sera également fourni lors de la réponse à la requête et permettra d'établir un lien entre la demande et la réponse.

  - Au niveau de la requête, le message envoyé vers le Registre national peut être signé avec un certificat utilisateur de type personne physique ou personne morale. C'est surtout lorsqu'une transaction de mise à jour transite par un intégrateur de services que cet usage est obligatoire. Les modifications des droits d'accès par service web nécessitent également la signature de la transaction.

3. En pratique, comment changer de génération de services web ?

Le changement du mode de fonctionnement SOAP vers REST et les nouvelles fonctionnalités peut être effectué à l'intervention de votre fournisseur informatique ou de votre propre service informatique.

Il n'y a pas d'intervention systématique nécessaire au niveau du Registre national, sauf en cas de changement de certificat ou de son chaînage.

Une documentation est fournie en annexe et comprend :

- le document RESTful API « in the Swagger Specification » des services REST du RN ;
- la documentation des changements apportés aux services existants (dont les nouvelles URL à utiliser) ;
- des exemples de test d'une API REST à l'aide de curl.

Un environnement de test est en place (8020). Il peut être utilisé pour la préparation de votre transition vers la 3e génération.

Parc Atrium
Rue des Colonies 11 T 02 518 21 16 helpdesk.belpic@rrn.fgov.be
1000 Bruxelles F 02 518 26 16 www.ibz.rrn.fgov.be

4. Qui contacter pour obtenir des renseignements supplémentaires ou de l'aide ?

Contacts techniques :

- El Habib El Meskioui, 02/518.21.33 - ElHAbib.ElMeskioui@rrn.fgov.be
- Paquito Bartiaux, 02/518.23.27 - Paquito.Bartiaux@rrn.fgov.be
- Ives Gobau, 02/518.21.05 - Ives.Gobau@rrn.fgov.be
- Nicolas Hermel, 02/518.22.39 - Nicolas.Hermel@rrn.fgov.be
- Helpdesk : 02/518.21.16 - Helpdesk.Belpic@rrn.fgov.be

Veuillez agréer, Mesdames, Messieurs, l'assurance de ma considération distinguée.

Jacques Wirtz
(Signature)
Signature numérique de Jacques Wirtz (Signature)
Date : 2020.11.05 12:31:11 +01'00'

Jacques Wirtz,
Directeur général

Parc Atrium
Rue des Colonies 11
1000 Bruxelles
T 02 518 21 16
F 02 518 26 16
helpdesk.belpic@rrn.fgov.be
www.ibz.rrn.fgov.be

RRN REST WS 3 génération

1. But du document

Ce document décrit la migration du RRN Web services SOAP 2ème génération vers RRN web services REST 3ème génération.

2. Accès aux web services

La sécurité des web services reste identique à la version SOAP 2ème génération, appliquée à la fois au niveau du transport et au niveau du message. Lors de l'interrogation du RN, les transactions entre l'application cliente et le service web sont sécurisées par le protocole TLS, l'application cliente est authentifiée par un certificat X.509.

3. Passer du modèle SOAP au modèle REST

la démarche est de projeter un ensemble d'opérations métiers sur une architecture orientée ressource. RRN web services existant est composés de deux web services SOAP, « SAML generator» et « Transaction executor» . Ces deux services contient chacun une méthode. Chacune d'elle possédant une signature et, éventuellement, des exceptions en cas d'erreur. Ces méthodes sont remplacées par des modèles d'interaction permettant d'implanter les fonctionnalités de l'application métier. La description de ces interactions est définie dans les documents OpenAPI fournis avec ce document. ces définition OpenAPI peut ensuite être utilisée par des outils de génération de documentation pour afficher l'API, des outils de génération de code pour générer des clients dans divers langages de programmation.

4. Signature des transactions de mise à jour

En plus des paramètres supplémentaires disponibles (voir openapi document) dans la version REST du RRN web services, cette version permet la signature du transaction de mise à jour. La signature des transactions de mise à jour est obligatoire pour tout client, ayant accès au RRN indirectement, via un intégrateur de services.

4.1. Transaction non-signée

- Le paramètre « txMessage » reste inchangé par rapport à la version SOAP,
- Le paramètre « signed » contient la valeur « 00 »
-

4.2. Transaction signée

- Le paramètre « txMessage » est remplacé par « signedMessage »,
- Le paramètre « signed » contient la valeur « 01 »,
- *La paramètre « signedMessage »* contiendra la concaténation des différentes parties de la JWS. Soient le header, le payload et la signature codés en Base 64 séparés par un point (appelée dans la norme RFC7515 : *the JWS Compact Serialization*[1]).
- Elément <header> du signedMessage
  - Paramètre indiquant le type d'algorithme utilisé pour la signature
    - alg : une des valeurs citées ci-dessus (RS256, RS384, RS512)
  - Paramètre nécessaire à la vérification des algorithmes de type RSA
    - x5c : Certificat ou chaine de certificats du signataire correspondant à la clé utilisée lors de la signature en une seule ligne.
- Elément <payload> du signedMessage
  - Ce paramètre reprend l'élément « txMessage » de la requête originale
  - Et contient un élément variable et unique pour chaque requête (cnonce : client number used once) à charge du client agissant avec l'intégrateur.
    Ex : ex : {"txMessage":"%XM25 RRN000---------00000C0----------00500011111000000----------NH",
    "cnonce":"6d8a3402-34f3-11e9-b210-d663bd873d93"}
  - le tout codé en Base 64.
- Elément <signature> du signedMessage
  - Contient le résultat de la signature de la concaténation du « header » avec le « payload » séparé par un point « . » par la méthode indiquée (alg) et avec la clé privée du certificat (x5c), le tout codé en Base 64.

5. URL des RRN REST web services

5.1. Environnement de test

SAML création :
https://www.webtcp20.rrn.fgov.be:5443/rrnws/rest/v3.0/assertions

Transaction exécution :
https://www.webtcp20.rrn.fgov.be:5443/rrnws/rest/v3.0/transactions

5.2. Environnement de production

SAML création :
https://www.rrnass.rrn.fgov.be:9920/rrnws/rest/v3.0/assertions

Transaction exécution :
https://www.rrnass.rrn.fgov.be:9920/rrnws/rest/v3.0/transactions

[1] https://www.rfc-editor.org/rfc/pdfrfc/rfc7515.txt.pdf (chapitre 7.1)

6. Exemples d'utilisation des Webservices 3 avec Curl

6.1. Paramètres de la commande curl

- CACERTFILE
  → path du fichier qui doit contenir le chainage complet des certificats CA du serveur
- saml-parameters.json
  → path du fichier qui doit contenir les différents paramètres donné pour obtenir l'assertion SAML (policy, certificat, requestId):
  Exemple :

```
{
    "policy":"CN_POLICY",
    "certificate":"BASE64_CERTIFICATE",
    "requestId":"12345678-1234-1234-1234-123456789012"
}
```

  BASE64_CERTIFICATE est remplacé par le certificat en format PEM, inclusive BEGIN et END limiteurs, et encodé en format BASE64.

- transaction-parameters.json
  → path du fichier qui doit contenir les différents paramètres de la transaction (txMessage, signed, signedMessage, rrnDestination, requestId) :
  Exemple :
  Non signé

```
{
    "txMessage":"%WS25 RRN0007C906206300000C07906011259105000100000021867906011259 1 F",
    "rrnDestination":"RRN_WS",
    "requestId":"12345678-1234-1234-1234-123456789012"
}
```

  Signé

```
{
        "signedMessage":"eyJhbGciOiJSUzI1NiIsIng1YyImcwSlcw.....",
        "signed":"01",
        "rrnDestination":"RRN_WS",
        "requestId":"12345678-1234-1234-1234-123456789012"
}
```

  Si le paramètre "signed" est manquant, le transaction est traité comme non signé.

- CLIENT_CERTIFICATE
  → path du fichier qui doit contenir la clé privée et le certificat publique au format PEM

6.2. Demande d’assertion SAML

curl --location --connect-timeout 15 --max-time 15 --cacert $CACERTFILE --data-binary @saml-parameters.json --header "Expect:" --header "Content-type: application/json" --cert $CLIENT_CERTIFICATE --url https://www.webtcp20.rrn.fgov.be:5443/rrnwa/rest/v3.0/assertions > samlAssertion8020

6.2.1. Edition du fichier généré (samlAssertion8020)

Le fichier reçu en réponse est au format JSON, et contient deux variables, "assertion" et "requestId".
Nous ne devons garder que l’assertion SAML seul (sans le nom du paramètre)et ajouter en début du fichier « Bearer » , pour une utilisation ultérieure lorsque nous exécuterons la transaction.

6.3. Exécution de transaction

curl --location --connect-timeout 15 --max-time 15 --cacert $CACERTFILE --header "Content-type: application/json" --header "Authorization: $(cat samlAssertion8020)" --data-binary @transaction-parameters.json --cert $CLIENT_CERTIFICATE --url https://www.webtcp20.rrn.fgov.be:5443/rrnws/rest/v3.0/transactions

```
openapi: 3.0.0
info:
    description: ""
    version: "3.0.0"
    title: "Swagger RRN REST SAML generator."
servers:
  - url: https://www.webtcp20.rrn.fgov.be:5443/rrnws/rest/v3.0
    description:  Main (Test Env 8020) server
paths:
  /transactions:
    post:
      summary: "Execute a given transaction."
      description: ""
      security:
        - bearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                requestId:
                  type: string
                txMessage:
                  type: string
                signedMessage:
                  type: string
                signed:
                  type: string
                rrnDestination:
                  type: string
                options:
                  type: string
                txOutputMessage:
                  type: string
                  description: for futur use.
      responses:
        '200':
          description: A Transaction response
          content:
            application/json:
              schema:
                type: object
                properties:
                  requestId:
                    type: string
                    description: The request ID.
                  transactionResponse:
                    type: string
                    description: The transaction response.
                  signed:
                    type: string
                    description: The flag signed/unsigned request.
                  rrnDestination::
                    type: string
```

```
                    description: The RRN Destiniation given in the
request.
                  options:
                    type: string
                    description: Options as given in the request.
                  txOutputMessage:
                    type: string
                    description: for futur use.

# 1) Define the security scheme type (HTTP bearer)
components:
  securitySchemes:
    bearerAuth:            # arbitrary name for the security scheme
      type: http
      scheme: bearer
      bearerFormat: Base64 SAML token    # optional, arbitrary value
for documentation purposes
# 2) Apply the security globally to all operations
security:
  - bearerAuth: []         # use the same name as above
```

```
openapi: 3.0.0
info:
    description: ""
    version: "3.0.0"
    title: "Swagger RRN REST SAML generator."
servers:
  - url: https://www.webtcp20.rrn.fgov.be:5443/rrnws/rest/v3.0
    description:  Main (Test Env 8020) server

paths:
  /assertions:
    post:
      summary: "Creates a token SAML."
      description: ""
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                requestId:
                  type: string
                policy:
                  type: string
                certificate:
                  type: string
      responses:
        '200':
          description: A SAML token
          content:
            application/json:
              schema:
                type: object
                properties:
                  requestId:
                    type: string
                    description: The request ID.
                  assertion:
                    type: string
                    description: The SAML token.
```