INFORMATIE VLAANDEREN

/// Rapport

MAGDA TOKEN-BASED AUTHENTICATIE

Handleiding voor MAGDA bronnen

Versie /// 1.2

Publicatiedatum /// 09/04/2020

Vlaamse overheid

Vlaanderen is verbonden

www.vlaanderen.be/informatievlaanderen

Auteur: Maxime Durenne

Datum aanmaak: 04 maart 2020

Datum afdruk: 27 april 2020

Documenthistoriek:

Versie	Opmerking	Datum	Auteur	Status
0.9	Opmaak document	04/03/2020	Maxime Durenne	Intern
1.0	Review Lieven en Kelly	05/03/2020	Kelly B., Lieven V.	Ready
1.1	Code voorbeelden en verder invullen over HTTP Signature	25/03/2020	Maxime Durenne	Intern
1.2	Review 6 en 9 april	09/04/2020	Lieven V	Intern

Informatie Vlaanderen

Havenlaan 88, 1000 Brussel
+32 (0)2 553 72 02

Koningin Maria Hendrikaplein 70, 9000 Gent
+32 (0)9 276 15 00

informatie.vlaanderen@vlaanderen.be

INHOUD

1 INLEIDING

Het aanbod van MAGDA werd begin 2020 uitgebreid met REST diensten. Voor deze REST diensten wordt gebruik gemaakt van een token-gebaseerd authenticatie systeem.

Token-gebaseerde authenticatie laat toe om authenticatie informatie naadloos te laten doorstromen van het ene systeem naar een andere. Ook multi-partijen scenario's kunnen hiermee gemakkelijker ondersteund worden. Bijvoorbeeld een scenario met een resource server (MAGDA) die informatie (of resources) ter beschikking stelt, een opdrachtgever (de entiteit die de recht heeft om de informatie te raadplegen) en een leverancier (de entiteit die de opvraging uitvoert in naam van de opdrachtgever).

2 DEFINITIE

Een token is een stuk informatie dat kan gebruikt worden om een identiteit en/of een recht te representeren.

Tokens kunnen ofwel onleesbaar (*opaque)* of *leesbaar* zijn.

Opaque tokens zijn een niet interpreteerbaar, ruw stuk informatie. Om ze te gebruiken is er altijd een derde partij (meestal de autorisatie provider) nodig om te weten als het token geldig is of aan welke toegang of identiteit het token is gelinkt.

Leesbare tokens zijn tokens die informatie bevatten over identiteit en/of autorisatie. JSON Web Tokens (JWTs) zijn leesbare tokens.

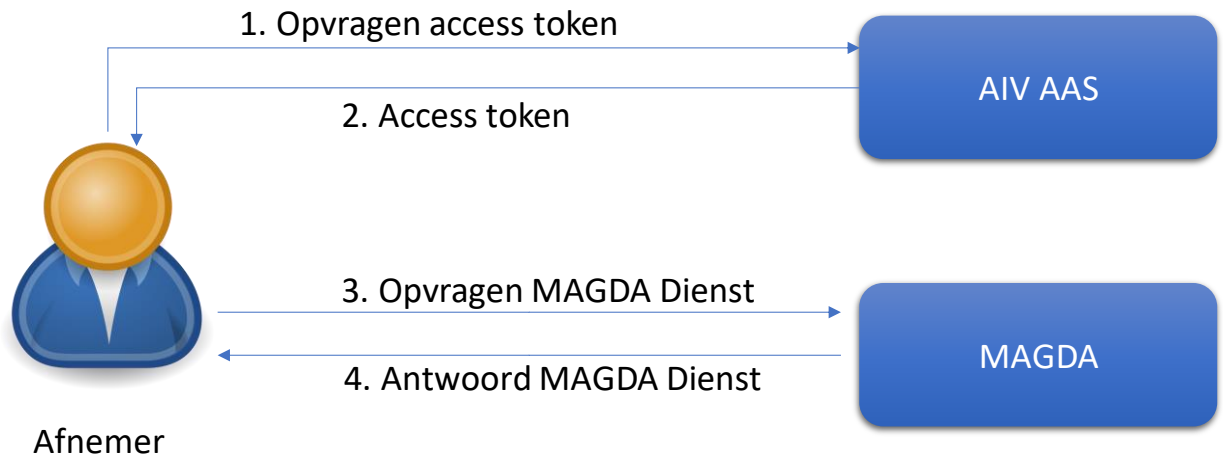
3 CONTEXT

MAGDA maakt gebruik van de AIV Authenticatie en Autorisatie Server (AAS) voor het authentiseren en autoriseren van afnemers in het geval van token authenticatie.

Werken met tokens omvat twee fases:

1) Het krijgen van een access token van een authenticatie en autorisatie server
2) Het gebruiken van een access token om een oproep naar een resource server te doen (bv een MAGDA dienst)

De access tokens die uitgekeerd worden door de AIV AAS zijn *opaque* tokens. Tijdens het proces om ze te krijgen wordt er gebruik gemaakt van een JWT (aanvraag token).



Stappen 1 en 2 maken deel uit fase 1: het krijgen van een access token.
Stappen 3 en 4 maken deel uit fase 2: het opvragen van MAGDA resource / diensten
Magda is ook de “resource server”.

4 FASE 1: HET KRIJGEN VAN EEN ACCESS TOKEN

4.1 AUTHENTICATIE EN AUTORISATIE AANVRAAG AANMAKEN

De afnemer stuurt een request naar de AIV AAS om een access token te krijgen.
Deze request moet een aanvraag token bevatten. Dit aanvraag token is een JWT token.
Dit token wordt getekend door een VO DCB certificaat met een door MAGDA erkende Common Name. Deze Common Name krijgt de afnemer van MAGDA tijdens het aansluitingsproces.
Er zijn twee AA servers, een voor TNI en een andere voor productie:

- TNI: https://beta.oauth.vlaanderen.be/authorization/ws/oauth/v2/token
- PROD: https://oauth.vlaanderen.be/authorization/ws/oauth/v2/token

4.1.1 Aanvraag JWT token aanmaken

Het JWT aanvraag token moet het volgende bevatten:

“iss” issuer eigenschap	de client id nummer die aan de afnemer gecommuniceerd is na de accounts aangemaakt zijn
“sub” subject eigenschap	moet ook met de client id ingevuld worden
“aud” eigenschap	audience van de request, moet altijd de URI van het token endpoint zijn, zie hierboven gemeeld lijst.
“exp” expiry eigenschap	het moment vanaf wanneer het token niet meer geldig zal zijn
“iat” issued at eigenschap	het moment waarop het token is aangemaakt
“jti” token id	een uniek nummer voor de aanvraag. (om “replay attacks” voort te komen)

Voorbeeld:

```
{
	"iss": "3318",
	"sub": "3318",
	"aud": "https://beta.oauth.vlaanderen.be/authorization/ws/oauth/v2/token",
	"exp": "1587979731",
	"iat": "1587979431",
	"jti": "19017aeb-3673-4f04-8ef7-e0756b80a667"
}
```

4.1.2 Tekenen van het JWT token

Het JWT aanvraag token voor de access token aanvraag moet getekend zijn.

De AIV AAS ondersteunt de “x5c”, “x5u”, “jwk”, “jku” eigenschappen niet. Dit betekent dat het certificaat dat gebruikt wordt om te tekenen, niet bij het token mag zitten. Dat certificaat moet op voorhand gecommuniceerd worden aan het MAGDA team zodat deze kan gebruikt worden voor authenticatie.

4.1.3 Aanvraag HTTP request aanmaken

De request moet een HTTP POST request zijn met x-www-form-urlencoded parameters. De volgende parameters moeten ingevuld worden:

- “grant_type”: “client_crendetials”
- “scope”: moet ingevuld worden met een lijst van scopes waaraan het access token toegang gaat geven. Er wordt een scope per dienst gebruikt. Dus voor elke MAGDA dienst die men wil aanroepen, moet hier de nodige scope aangevraagd worden. “scopes” worden door middel van een spatie van elkaar afgescheiden.
- “client_assertion_type”: “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”
- “client_assertion”: het JWT token beschreven in het vorige punt

Voorbeeld:

```
POST https://beta.oauth.vlaanderen.be/authorization/ws/oauth/v2/token
Content-Type: application/x-www-form-urlencoded
User-Agent: PostmanRuntime/7.22.0
Accept: */*
Cache-Control: no-cache
Host: beta.oauth.vlaanderen.be
Accept-Encoding: gzip, deflate, br
Content-Length: 855
Connection: keep-alive

grant_type: "client_credentials"
```

```
scope: "msg_pubMsg_v1_P msg_mailbox_v1_P msg_searchMsg_v1_P"
client_assertion_type: "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
client_assertion: "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiIzMzE4Iiwic3ViIjo
iMzMxOCIsInNjb3BlIjoibXNnX3B1Yk1zZ192MV9QIiwiYXVkIjoiaHR0cHM6Ly9iZXRhLm9hdXRoLnZsYW
FuZGVyZW4uYmUvYXVfaG9yaXphdGlvbi93cy9vYXV0aC92Mi90b2tlbiIsImV4cCI6IjE1ODMzMzQ5MTMiL
CJpYXQiOiIxNTgzMzM0NjEzIiwianRpIjoiNzBiZGUzNzgtOWIxYi00MTljLThmNGEtZWYzYTRkMTU1ZjJh
In0.qub7FvsKPB7qQzfYXGe1saGe5YgDPnwEp9ATnSL9YTCyDMFMYoN0rHusXmaZ_zrq4-
NaVuktEmimeVCN2TL3R4O5e6jHcNk9C7IwMfFFu2NTnitSS2j_Y7zU2T0vEplKaJyanJ7UryCfsSe7tYI-
mJzYObCOzrT1MO4hGCpoXvzYTWrM8SqclHIHrIieMtCKDe7a1HbaIngcRnizBDBG-vSskWtf4-
tFurkRTEwL9v3D-wiPk2BYjcnupoj1C15cUeSIF_s2LNwoBoqhIFEmx-
LLSSQXnwit588XANvkbpBpelxnUep3NgjazEAAq1SCEDWbcXrpt0ZBo3rQz3jScg"
```

4.2 ANTWOORD VAN AAS

Als de authenticatie aanvraag correct verlopen is, krijgt de aanvrager een JSON payload terug met het access token.

```
{
    "access_token": "POtyWHuQd94pk6AtHbew3B==",
    "scope": "msg_pubMsg_v1_P msg_mailbox_v1_P msg_searchMsg_v1_P",
    "expires_in": 57599,
    "token_type": "Bearer"
}
```

De eigenschap “access_token” bevat het token voor de call naar elke MAGDA REST dienst (resource).

Access tokens zijn momenteel uitgereikt met een levensduur van 8 uren. Dit betekent dat na 8 uren het token ongeldig zal worden en er zal door de afnemer een nieuw token moet aangevraagd worden om verder te kunnen. Dit gebeurt best vooraleer het token ongeldig wordt om de continu gebruik van het token mogelijk te maken.

4.3 CODE VOORBEELD

In de volgende voorbeelden wordt er van Apache Http Components (HttpClient) gebruik gemaakt om de HTTP Call uit te voeren en van jose4j voor het aanmaken van de JWT. Andere libraries zullen op een gelijkaardige manier werken.

Het bouwen van de getekend JWT met jose4j:

```
private static String buildClientAssertion(String clientId, PrivateKey key) throws
JoseException {
        JwtClaims claims = new JwtClaims();
        claims.setIssuer(clientId);
        claims.setAudience(AIP_AUDIENCE);
        claims.setExpirationTimeMinutesInTheFuture(2.0f);
```

//

```
        claims.setIssuedAtToNow();
        claims.setGeneratedJwtId();
        claims.setSubject(clientId);

        JsonWebSignature signature = new JsonWebSignature();
        signature.setPayload(claims.toJson());
        signature.setKey(key);
        signature.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);
        signature.setContentTypeHeaderValue("JWT");

        return signature.getCompactSerialization();
}
```

Voor het aanmaken van een HttpClient http request:

```
private static HttpPost buildAccessTokenRequest(String scopes, String signedJWT)
throws UnsupportedEncodingException {
        HttpPost post = new HttpPost(AIP_TOKEN_PROVIDER);

        ArrayList<BasicNameValuePair> retVal = new ArrayList<BasicNameValuePair>();

        retVal.add(new BasicNameValuePair("grant_type", "client_credentials"));
        retVal.add(new BasicNameValuePair("scope", scopes));
        retVal.add(new BasicNameValuePair("client_assertion_type",
"urn:ietf:params:oauth:client-assertion-type:jwt-bearer"));
        retVal.add(new BasicNameValuePair("client_assertion", signedJWT));

        post.setEntity(new UrlEncodedFormEntity(retVal));

        post.setHeader("Accept", "application/json");

        return post;
}
```

Daarna kan de request verstuurd worden:

```
public static String getAccessToken(PrivateKey authenticationKey, String scopes[])
throws JsonParseException, JsonMappingException, UnsupportedOperationException,
IOException, JoseException {
        HttpClient c = HttpClientBuilder.create().build();

        String scopeStr = String.join(" ", scopes);
        String signedJWT = buildClientAssertion("3318", authenticationKey);

        HttpPost target = buildAccessTokenRequest(scopeStr, signedJWT);
        HttpResponse resp = c.execute(target);

        assertEquals(200, resp.getStatusLine().getStatusCode());
        ObjectMapper mapper = new ObjectMapper();
        LiveTest.AccessTokenResponse token =
mapper.readValue(resp.getEntity().getContent(),
LiveTest.AccessTokenResponse.class);

        return token.access_token;
}
```

//

Als het request correct is, antwoord de AAS met een antwoord zoals beschreven in 4.2

5 FASE 2: HET OPVRAGEN VAN MAGDA RESOURCES

5.1 AUTHENTICATIE

Voor elke MAGDA resource "request" is een geldig access token nodig, dat de afnemer in Fase 1 verkregen heeft.

Het access token wordt meegegeven in de "Authorization" HTTP parameter. De token wordt voorafgegaan door de prefix "Bearer".

```
POST https://magda-api.vlaanderen.be/api/v1/messages/publishMessage
Authorization: Bearer POtyWHuQd94pk6AtHbew3B==
```

De Magda resource server adressen zijn de volgende:

- Test: https://informatie-vlaanderen-test.apigee.net
- TNI: https://informatie-vlaanderen-tni.apigee.net
- Productie: https://informatie-vlaanderen-prod.apigee.net

5.2 REQUEST SIGNING

Om bericht integriteit te garanderen over meerdere technische en functionele integratie lagen, wordt met HTTP signing gewerkt. HTTP Signing is een IETF standard.

HTTP signing wordt goed ondersteund in Java via de Apache CXF library.

Deze HTTP signing gebruikt het VO DCB certificaat met een door MAGDA erkende Common Name.

Meer info over signing in het volgende hoofdstuk **7 Message Signing**.

5.3 RESPONSE SIGNING

MAGDA tekent alle haar antwoorden met het eigen MAGDA certificaat.

6 GEBRUIK VAN HET VO DCB CERTIFICAAT

Een korte recap van het gebruik van het VO DCB certificaat met als CN de URI verkregen bij de aansluiting aan de MAGDA dienst.

- Bij de aanvraag (fase 1): tekenen van het JWT aanvraag token
- Bij de toegang tot de MAGDA dienst (fase 2): tekenen van de message van de request

7 MESSAGE SIGNING

Message Signing verwijst naar het tekenen van berichten om hun integriteit te garanderen. Het tekenen van het bericht met een certificaat helpt de oorsprong van het bericht te identificeren. Deze message signing met certificaat is enerzijds nodig als beveiliging bovenop het TLS kanaal. Anderzijds biedt dit de mogelijkheid om de HTTP berichten door meerdere kanalen te laten gaan zonder het bewijs van de identiteit van de bron te verliezen met gegarandeerde integriteit.

Voor het tekenen van HTTP berichten bestaat er een IETF draft standaard (12de iteratie): https://tools.ietf.org/id/draft-cavage-http-signatures-12.html

Omdat REST geen standaardisatie heeft voor payload signing, is er geen officieel gestandaardiseerde implementatie (library) van de IETF draft HTTP handtekening specificatie. Er zijn daarentegen wel een aantal “open source” implementaties.

7.1 MESSAGE SIGNING PRINCIPES

Het principe is als volgt:

- De verplichte kenmerken van het HTTP bericht worden verzameld:
  - Tijdstip van het request (“Date” header)
  - Digest (hash) van het bericht (“Digest” header), zoals in IETF draft-ietf-httpbis-digest-headers-00 gedefinieerd. De “digest” moet op voorhand berekent zijn. En moet minstens sha-256 gebruiken. Sha-1 wordt niet toegelaten.
  - Resource identificatie (http verb + target path), wordt hieronder “(request-target)” vernoemd.
  - De publieke sleutel die gebruikt mag worden om de signature te valideren.
- De kenmerken worden met een private sleutel getekend.
- De handtekening wordt in de request als HTTP header toegevoegd (“signature” header), samen met een rekeningmethode die beschrijft welke kenmerken worden getekend, en in welke volgorde.

Bijvoorbeeld:

```
   POST /foo HTTP/1.1
   Host: example.org
   Date: Tue, 07 Jun 2014 20:51:35 GMT
   Content-Type: application/json
   Digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
   Content-Length: 18
   Signature: keyId="rsa-key-1",algorithm="rsa-sha256",
   headers="(request-target) date digest signature-public-key",
   signature="XX…XXX"
   signature-public-key: {"kty":"RSA","kid":"rsa-key-1","n":"r3-
......_D4iw6fDFw2mEQ", "e":"AQAB","x5c":["MIIFtjCCB.........="]}

   {"hello": "world"}
```

De “Signature” HTTP header heeft de volgende parameters:

- **keyId:** de naam van de sleutel die gebruikt was. Dit is handig als er meerdere sleutels gebruikt kunnen worden.

//

- **algorithm:** een beschrijving van het algoritme (tekening) dat gebruikt is geweest. Dit is sinds draft 11 van de specificatie verouderd. Algoritmen moeten afgeleid worden van de meegegeven 'keyId'. Minimum eisen: rsa 2048-bit, sha-256 (beter is 512). (b.v. rsa-sha256 of rsa-sha512).
- **headers**: de lijst van de kenmerken die gebruikt zijn om de signing string op te maken. Deze zijn altijd HTTP headers, behalve voor "(request-target)" die de target url representeert.
- **signature:** de handtekening zelf, als Base64 gecodeerd.

In de context van MAGDA, zullen de asymmetrische sleutels van een VO-DCB Signing certificaat moeten komen (voor entiteiten die niet onder VODCB scope vallen zullen ze uitzonderlijk op voorhand moeten gecommuniceerd worden). Op die manier zullen de sleutels duidelijk erkend worden en zal ook de mogelijkheid bestaan om een revoke van de certificaat te doen. De geldigheid van de certificaat wordt ook gecontroleerd.

Alle requests van MAGDA afnemers moeten met dit systeem tekenen, met een keyId die het Magda certificaat identificeert. Bijvoorbeeld: "AfnemerXCertificaat" (de naam van de sleutel is niet belangrijk, zolang die coherent blijft tussen de sleutel en de signature).

Om toe te laten deze verificatie te doen zonder het certificaat op voorhand te moeten uitwisselen, zal het certificaat mee in de request moeten staan.

Het antwoord van MAGDA wordt met het MAGDA certificaat getekend. Bijvoorbeeld: keyId="magda-response-signing-key".

7.1.1 Java Code voorbeeld

Met Apache HttpClient component en tomitribe.http-signatures-java:

```
public static void httpSign(HttpEntityEnclosingRequestBase request, PrivateKey
signingKey, X509Certificate certificate) throws UnsupportedOperationException,
NoSuchAlgorithmException, IOException, JoseException {
	computeSHA256Digest(request);
	request.addHeader("signature-public-key", wrapCertificateInJWK(certificate,
"key1"));

	final Signature signature = new Signature("key1", "rsa-sha256", null,
"(request-target)", "date", "digest", "signature-public-key");

	final Signer signer = new Signer(signingKey, signature);
	final Signature signed = signer.sign(request.getMethod(),
request.getURI().getPath().toString(), extractHeadersToSign(request));

	request.addHeader("Signature", signatureToHeaderFormat(signed));
}


static String signatureToHeaderFormat(Signature s) {
	return "keyId=\"" + s.getKeyId() + '\"' +
		",algorithm=\"" + s.getAlgorithm().getPortableName() + '\"' +
		",headers=\"" + Join.join(" ", s.getHeaders()) + '\"' +
		",signature=\"" + s.getSignature() + '\"';
}


private static Map<String,String> extractHeadersToSign(AbstractHttpMessage
message) {
```

```
        HashMap<String,String> result = new HashMap<String,String>();

        result.put("date", message.getFirstHeader("date").getValue());
        result.put("digest", message.getFirstHeader("digest").getValue());
        result.put("signature-public-key", message.getFirstHeader("signature-public-key").getValue());

        return result;
}
```

Met Apache CXF

```
public Response getSourceResponse(String address) {
    MessageSigner messageSigner = new MessageSigner(keyId -> privateSignKey, "rsa-key-1");
    signatureFilter.setMessageSigner(messageSigner);
    signatureFilter.setDigestAlgorithmName("SHA-256");
    signatureFilter.setAddDigest(true);

    WebClient webClient = WebClient.create(address,
Collections.singletonList(signatureFilter));
    webClient.getConfiguration().getOutInterceptors().add(new TLSInterceptor());
    webClient.header("Content-Type", "application/json");
    webClient.header("Accept", "application/json");
    webClient.header("Date",
httpDateFormater.format(Calendar.getInstance().getTime());

    Response response = webClient.get();

    return response;
}
```

Met CXF is het zeer eenvoudig, het is genoeg om een “signature filter” toe te voegen aan de WebClient.

7.2 PUBLIEKE SLEUTEL VOOR VERIFICATIE VAN DE HANDTEKENING

MAGDA stelt een asymmetrisch signing protocol voor. Dit betekent dat er alleen nood is aan de publieke sleutel om de handtekening te valideren.

De gemakkelijkste manier om publieke sleutels te verspreiden zijn de X.509 Certificaten. Zij laten toe om de identiteit van de sleuteldrager te verzekeren.

Voor het tekenen van MAGDA request wordt er gevraagd om met VODCB uitgegeven certificaten te werken. Op deze manier zijn de certificaten automatisch door MAGDA vertrouwd en moeten zij niet *op voorhand* gekend zijn (door b.v. ze op voorhand te installeren in onze systemen). Daardoor is ook vernieuwing van het certificaat eenvoudiger.

Om de handtekening te valideren, wordt het certificaat mee gegeven met elke request. Dit gebeurt door een JSON Web Key (JWK) in de “**signature-public-key**” te steken.

Het JWK formaat laat toe om X509 Certificaten met de sleutel te sturen. In ons geval is het verplicht om ze in de JWK te steken (door de x5c eigenschap te gebruiken).

De JWK moet een “keyId” bevatten en die moet met de “keyId” van de signature overeenkomen. Dus als de handtekening verwijst naar een sleutel met id “magda” dan moet de JWK keyId ook “magda” zijn. Als een sleutel met overeenkomende id niet gevonden kan worden, gaat de signing validatie falen.

> ☛ *Opmerking: Zolang de berichten worden getekend met dezelfde sleutel, is de publieke sleutel ook altijd dezelfde. Dus de inhoud van de “signature-public-key” blijft ook constant.*

7.2.1 Java code voorbeeld

Dit voorbeeld gebruikt de Jose4j library.

```
public static String wrapCertificateInJWK(X509Certificate cert, String keyName)
throws JoseException {
	PublicJsonWebKey jwk =
			PublicJsonWebKey.Factory.newPublicJwk(cert.getPublicKey());

	jwk.setCertificateChain(cert);
	jwk.setKeyId(keyName);

	return jwk.toJson();
}
```

Dit functie geeft de inhoud terug die in het “signature-public-key” header moet staan

```
signature-public-key: {"kty":"RSA","kid":"key1","n":"r3-......_D4iw6fDFw2mEQ",
"e":"AQAB","x5c":["MIIFtjCCB.........="]}
```

8 BIJKOMENDE INFORMATIE

8.1 GEOSECURE INFORMATIE

Bijkomende informatie over de AIV GeoSecure AAS oplossing kan gevonden worden op:
https://beta.oauth.vlaanderen.be/authorization/Help/Api/ClientCredentialsGrant

8.2 HTTP MESSAGE SIGNING IMPLEMENTATIES

- Voor Java
  - Apache CXF: dit implementatie is door MAGDA getest en bevestigd te werken.
  - Tomitribe “http-signatures-java”: deze implementatie is door MAGDA getest en bevestigd dat deze correct werkt. Opmerking: opletten op validatie van “date” en “digest” headers: deze zit niet in de library en moet door de client gebeuren.
- In Javascript
  - Inspiratie zou kunnen genomen van dit Postman script, die momenteel alleen HMAC signing ondersteunt, maar zou kunnen uitgebreid worden om RSA signing te ondersteunen. Indien nodig kan MAGDA zo’n implementatie ter beschikking stellen.
- In C# (geen enkele van de volgende links zijn gereviewed, kwaliteit kan varieren)
  - https://github.com/karlbohlmark/http-signatures-csharp

//

- https://github.com/puckipedia/Kroeg/blob/master/Kroeg.Services/
- https://github.com/DavidLievrouw/HttpMessageSigning