

Meerdere signature (algoritmen) voor Signed Open Badges

JWS **Compact** Serialization vs JWS **JSON** Serialization

Waar te vinden in vorig 'Potentieel doelen overzicht'?

Te implementeren onderdelen in grote lijnen						
	Wel/Niet	Prioriteits advies	Must have	Nice to have	Non-goal	Bron
Signing met 1 signature mechanism (makkelijk te veranderen tegen de tijd dat QC een probleem wordt)	x	1	x			Surf
Secure key storage (hardware modules)	x	1	x			Rapport RR 7.3
Secure key generation (hardware module en/of anders)	x	1	x			Rapport RR 7.5
Timestamping (secure, trustless (onafhankelijk te raadplegen))	x	1	x			Nieuw
Validatie van signed open badge (met uitgever pub-key)	x	1	x			Surf
Meerdere signing keys (multisig) (docent + Examen Commissie bijv)	x	2		x		Rapport RR 7.4
Encryptie van signed open badge (met houder pub-key)	x	3		x		Surf
Ontvanger publickey/address implementatie (alternatief Eduld), handig voor revocation.	x	4		x		Nieuw
Meerdere signature mechanisms		999			x	Rapport RR 7.1
Re-signing van open badges		999			x	Rapport RR 7.2
Quantum resistant algoritme (waar nog geen consensus over is) implementeren.		999			x	Rapport RR 7.1

Welke doelen kan het gebruik van meerdere signature (algoritmen) bereiken voor Signed Open Badges(SOB)? (1/2)

Er zijn twee hoofdredenen waarom iemand zou kunnen kiezen voor implementatie van meerdere signature (algoritmen) in Signed Open Badges.

- **Om twee of meer verschillende uitgevers van een open badge te hebben.**
 - Dit kan ervoor zorgen dat de kans op frauduleus uitgeven van een badge kleiner wordt.
 - Dit resultaat kan echter ook behaald worden _zonder_ meerdere signature (algoritmen) in een signed open badge.
 - Zie: <https://crypto.stackexchange.com/questions/50448/schnorr-signatures-multisignature-support>

Welke doelen kan het gebruik van meerdere signature (algoritmen) bereiken voor Signed Open Badges(SOB)? (2/2)

Er zijn twee hoofdredenen waarom iemand zou kunnen kiezen voor implementatie van meerdere signature (algoritmen) in Signed Open Badges.

- **Om meer dan één signature algoritme te kunnen gebruiken voor het uitgeven van een SOB.**
 - Dit kan het moeilijker maken om de gecombineerde security* van een SOB (signature) te 'kraken'.
 - Een quantum computing resistant algoritme kan toegevoegd worden.**

*Het toevoegen van meerdere algoritmen in een SOB maakt het kraken moeilijker op een lineaire manier (QC resistentie niet meegenomen). Een terechte vraag is of dit de implementatie opgave waard is.

**Dit brengt nog wel verdere moeilijkheden met zich mee. Zie overige info.


Welke vormen van JWS Serialisation zijn er?

- **JWS Compact** (URL safe) Serialisation

- Base64 encoded
- Header
- Payload
- Signature
- Onderdeel van Open Badge standaard (alleen bijna niet gebruikt en onderhouden)

- **JWS JSON** (General) Serialisation

- JSON format
- Header protected (per signature)
- Header unprotected (per signature) (optioneel)
- Signatures
- Payload
- Geen onderdeel van Open Badge standaard



Dit format is nodig voor meerdere signature (algoritmes)

Het format van JWS Compact Serialisation

Header

`{"alg": "ES256"}` ← **signature algo**

Payload

```
{ "@context": [  
  "https://w3id.org/openbadges/v2",  
  { "etc": "etc" } ] }
```

 ← **Assertion + Timestamping proof**

Volledige signature (JWS Compact Serialisation)

`eyJhbGciOiJIUzI1NiJ9` ← **header (base64 url encoded)**

`.eyJpc3MiOiJqb2UiLA0KICJleHAiOiEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlIjMnNvbS9pc19yb290Ijp0cnVlfQ` ← **payload**

`.DtEhU3ljbEg8L38VWAFUAqOyKAM6-Xx-F4GawxaepmXFCgfTjDxw5djxLa8ISISApMwQxfKTUJqPP3-Kg6NU1Q` ← **signature**

Het format van JWS JSON Serialisation

General JWS JSON Serialization syntax (onderdeel van RFC 5741):

```
{  
  "payload": "<payload contents>", ← (base64 encoded) assertion zit hier  
  "signatures": [  
    {  
      "protected": "<integrity-protected header 1 contents>", ← signature algo  
      "header": "<non-integrity-protected header 1 contents>", ← optionele info  
      "signature": "<signature 1 contents>", ← signature  
    },  
    ...  
    {  
      "protected": "<integrity-protected header N contents>",  
      "header": "<non-integrity-protected header N contents>",  
      "signature": "<signature N contents>"}  
  ]  
}
```

<http://self-issued.info/docs/draft-ietf-jose-json-web-signature.html#JSComplete>

Hoe gaat 'baking' van een Signed Open badge?

Bij het baking proces van open badges wordt een stukje data in een afbeelding verwerkt (via read en write streams).

In het geval van signed open badges wordt simpelweg de JWS signature op diezelfde manier 'gebakken' in een afbeelding.

Het is dan aan de validator code van de raadpleger of die kan omgaan met extractie van de signature uit de gebakken open badge.

Voor een overzicht hoe JWS signature verwerkt worden in Timestamped Signed Open Badges → [Diagram](#) van digitale objecten voor Badge creatie.

Hoe gaat validatie van een Signed Open badge?

De IMS Global Open Badge documentatie geeft het volgende aan voor verificatie van de inhoud van een SOB.

1. *Base64 decode the JWS payload. This will be a JSON string representation of the badge assertion.*
2. *Parse the decoded JWS body into a JSON object. If the parsing operation fails, assertion MUST be treated as invalid.*
3. *Perform data validation on the Assertion and the (embedded or linked) BadgeClass and issuer Profile, as well as any other relevant present data.*
4. *Determine the correct signing key id(s) to test against the signature from the valid issuer Profile and Assertion VerificationObject. Do not trust a key that is not properly linked from the issuer Profile.*
5. *Perform an HTTP GET request on the trusted keys. If it is impossible to get a usable public key, the Assertion cannot be verified.*
6. *With each trusted public key, perform a JWS verification on the JWS object. If the verification fails, assertion MUST be treated as invalid.*
7. *Retrieve the revocation list from the Issuer Profile object if present and ensure the id of the badge does not appear in the revokedAssertions list. Legacy v1.x badges that lack a id property may be identified in this list by their uid.*
8. *If the above steps pass, assertion may be treated as valid.*

Welke moeilijkheden zijn er om JWS JSON Serialisation te implementeren?

Ergens in de **Assertion** van de open badge moet worden aangegeven **hoeveel signatures** er **verwacht** worden. **Dit vereist een extensie.**

De Open Badge standaard **kan** momenteel **niet omgaan met JWS JSON Serialisation**. Dit **vereist een validator update bij alle verifiers** van Open Badges én dus een **consensus** onder Open Badge **stakeholders**.

[^] (In de IMS Global documentatie wordt wel gesuggereerd dat er mogelijkheid tot inbreng van andere signature methoden is).

En Quantum Computing Resistente algoritmen dan?

Zoals eerder aangegeven is er nog onzekerheid over welke post-qc algoritmen daadwerkelijk in gebruik genomen gaan worden.

Daarnaast moeten deze in de praktijk getest worden om potentiële 'bugs' aan het licht te laten komen.

Alle post-qc algoritmen die in de running zijn implementeren in een signed open badge is eigenlijk geen optie (aangezien het veel, complex en continu veranderend is).

Advies voor **JWS Compact** of **JWS JSON** Serialisation

Gebruik JWS Compact Serialisation,

Dit biedt:

- Huidige Open Badge implementaties mogelijkheid tot valideren van signature.
 - Geen aanpassing van extensie voor Issuer pubkeys (t.b.v. signing).
 - Geen aanpassing van extensie voor Assertion signing key count.
 - Geen aanpassing validatie stappen door Open Badge gebruikers.
- Alsnog post-QC security omdat Signed Open Badge ook een Timestamp krijgt.
- Bij gebruik van EdDSA of ECDSA als algoritme kan ook multi-signature mogelijk zijn.