

# Enveloped JSON Signatures

*Although XML Signatures are extremely flexible they come at a price: limited interoperability and mobile platform support. The case for XML has also been considerably weakened by REST which more or less has replaced SOAP for web-based systems used by for example Google and Facebook. In REST-based systems the returned data is usually in JSON format.*

## Converting to JSON

Due to the above I felt a need to convert the currently XML-based KeyGen2 system to JSON. However, there is currently no counterpart to XML DSig's enveloped signatures which made me develop such a system. It turned out that less than 3,000 lines of Java code were required to *Encode/Decode* and *Sign/Verify* JSON data:

<https://code.google.com/p/openkeystore/source/browse/#svn%2Flibrary%2Ftrunk%2Fsrc%2Forg%2Fwebpki%2Fjson>

I can now safely retire my 200,000+ lines Android port of Xerces ☺

Things that make JSON signatures simpler than XML DSig include:

- No attributes that must be sorted
- No namespaces
- No defaults
- No SOAP envelopes
- No WS-Security framework

Obviously there are complex systems that live or die by the use of XML DSig and XML Schema but KeyGen2 is not one of them...

The JSON parser mentioned also does a pretty good job for supporting conformance checking with intended messages though registered message types as well through strict property order and reference checks.

## Sample Signature

```
{
  "TestSignatures":
  {
    "@jmsns": "http://example.com/signature",
    "Now": "2013-08-29T08:04:07+02:00",
    "HRT":
    {
      "RTl": "67",
      "YT":
      {
        "HTL": "656756#",
        "INTEGER": -689,
        "Fantastic": false
      },
      "er": "33"
    },
    "ARR": [],
    "BARR":
    [{
      "HTL": "656756#",
      "INTEGER": -689,
      "Fantastic": true
    },
    {
      "HTL": "656756#",
```





## Other JSON Signature Solutions

The IETF JOSE WG has defined a JSON signature scheme called JWS. The reason why I haven't adopted JWS for KeyGen2 is because it is based on in-line signatures using Base64-encoded payloads. Although certainly working it disrupts readability which makes the switch from XML to JSON unnecessary painful for schemes where the *message* is the core and a signature only is there to vouch for the message's authenticity. The following shows how a JWS-based conversion of the former message would look like:

```
{
  "message": "dTzJcZgb...SvyF3ZYV.NH1L...cQOFsMoQ8oU.0GA1UdDg...QWBBRaQnES"
}
```

## Author

[anders.rundgren.net@gmail.com](mailto:anders.rundgren.net@gmail.com)

2013-08-29