DSS core refresh

While celebrating DSS being an official standard for 10 years now there is a need to brush up the documents and schemas to keep up with the latest development, drop outdated elements and address a small number of bugs. The design goals for the refresh of DSS/X scheme are:

- Simplify the core schema
- Support other transport formats (especially JSON / REST)
- Focus on easy use and implementation in common usage environment (Java, JavaScript, ...)
- Drop / workaround XML specifics (like e.g. xs:any, mixed content)
- Drop elements not used (any more)

Simplifcations of the core schema

- Unified the way documents can be represented by dropping InlineXML, Base64XML and EscapedXML. Base64Data is the primary way to transfer documents.
- Drop the generic 'Other' element of type dss:AnyType from the dss:InputDocuments
- Unify the signatures representation using Base64Signature as the primary way to represent timestamps and signatures
- Drop of the Timestamp structure. It is note referenced anymore.
- Drop of Request's 'Profile' attribute
- Renamed OptionalInput 'AdditionalProfile' to 'Profiles'
- Added 'AppliedProfiles' to OptionalInputs
- Added id/idref pairs to enable de-duplication of binary values (e.g. CRLs in the verification report)
- Define cardinality of OptionalInputs and OptionalOutputs child elements explicitly
- Introduce specific OptionalInputs and OptionalOutputs for sign and verify holding explicit enumeration of valid elements including cardinality. It's difficult to extract the relevant information from the core description document. So please review in detail.
- Change from flag-typed elements to Boolean types
- Dropped 'AttachmentReference' in favour of a more general reference attribute
- Replaced the dss:AnyType by xades:AnyType
- Use of xs:any replaced by xs:base64Binary and a MIME type attribute for non-XML bindings
- Minimize reference to other schemes:
    - Replaced ds:KeyInfoType with a more specific structure in dss:KeySelector using a X509Digest as a replacement of the deprecated X509IssuerSerial

- Replaced ds:KeyInfoType with a more specific structure based on dss:KeySelector as the type of dss:AdditionalKeyInfo

Support for other transport formats

In addition to the additional bindings it is intended to have good implementation support by the DSS/X core. A promising approach is to use e.g. Java's existing infrastructure for different bindings. Therefore a Java object model is derived from the existing XML schema . The object model will be used to apply other bindings (e.g. JSON) and derive specific binding schema artefacts. This approach has two advantages:

- The TC has to maintain just one schema, the other ones can be derived
- Implementability is ensured

The outline for the technical validation looks like:

- Use the given DSS 1.0 schema files for core and profiles and the referenced schemes
- Transform the files using XSL stylesheets. One advantage of this approach is the traceability of the applied changes
- Focus on Base64 as the most flexible way to transport documents and signatures
- Replace xs:any by replacing it with an enumeration of possible types. Use base64 blobs as a fallback
- Rearrange sequences and choices to produce a strongly typed object model (e.g. avoid List<Serializable>). So the need for manual de-serializing of objects is minimized.
- Use Java's JAXB schema compiler to derive an universal object model serializable to XML and JSON. And presumably to other formats, e.g. YAML
- Derive the JSON schemes from the Request and Response objects
- Have a set of test cases that ensures the complete created object tree can be serialized and deserialized without affecting the content
- Use the proven XML schema as the base for a committee draft

Support for extensions introduced by profiles

The approach to avoid the use of xs:any in the core (especially within OptionalInputs and OptionalOutputs) has a negative effect on the extensibility. New types introduced by profiles will be rejected by the core schema. But this advantage just affects the interface definition level. The implementation MUST be able to handle the full set of expected elements. Moreover, in the case of xs:any use, the schema compiler (e.g. JAXB) leaves the implementor with the task of handling generic types manually. Inserting the relevant type definitions for the specific implementation allows more expressive schemes and implementation support:

- The use of xs:any is convenient for the writers of specifications
- It doesn't help to describe a **real** service and it's capabilities
- The support for available profiles is **not** visible at the WSDL endpoint
- The expected structure of the xs:any inhabitants is hidden in the schema. For dynamic web service consumer like SOAP UI or BPMN clients this degrades usability
- Even in type-safe environments (like Java and C++) the handling of profile-specific extensions requires a 'dynamic' implementation style (check for the occurrence of all possible types and do specific down-casts)
- Tricky to ensure proper testing coverage
- Documentation-only requirements (e.g. cardinality) can easily get lost with an xs:any approach. This may cause interoperability issues

To support the dynamic creation schema files corresponding the the supported profiles a stylesheet (or a website) will be provided to create the schema aware of the set of supported profiles.