Compound Request/Response Profile of the OASIS Digital Signature Service (DSS)

Working Draft 01, 27 December 2005

Document identifier:

Location:

http://www.oasis-open.org/committees/dss

Editor:

Carlos González-Cadenas, netfocus

Contributors:

Marta Cruellas, CATCert

Francesc Oliveras, CATCert

Ignacio Alamillo, CATCert

Abstract:

This document defines XML request/response protocols for requesting compound DSS-based operations to a server.

Status:

This is a **Working Draft** produced by the OASIS Digital Signature Service Technical Committee. Committee members should send comments on this draft to dss@lists.oasis-open.org.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Digital Signature Service TC web page at http://www.oasis-open.org/committees/dss/ipr.php.

Table of Contents

1 Introduction

1.1 Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 **[RFC 2119]**. These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

This specification uses the following typographical conventions in text: `<CompoundProfileElement>`, `<ns:ForeignElement>`, `Attribute`, **Datatype**, `OtherCode`.

```
Listings of Compound Profile schemas appear like this.
```

1.2 Schema Organization and Namespaces

The structures described in this specification are contained in the schema file **[Compound-XSD]**. All schema listings in the current document are excerpts from the schema file. In the case of a disagreement between the schema file and this document, the schema file takes precedence.

This schema is associated with the following XML namespace:

```
urn:oasis:names:tc:dss:1.0:profiles:compound
```

If a future version of this specification is needed, it will use a different namespace.

Conventional XML namespace prefixes are used in the schema:

- The prefix `dss:` stands for the DSS core namespace **[Core-XSD]**.
- The prefix `ds:` stands for the W3C XML Signature namespace **[XMLSig]**.
- The prefix `xs:` stands for the W3C XML Schema namespace **[Schema1]**.
- The prefix compound: or no prefix defaults to the namespace of the present document.

2 Profile Features

2.1 Identifier

urn:oasis:names:tc:dss:1.0:profiles:compound

2.2 Scope

At the moment, with the capabilities included in the DSS Core protocols **[DSSCore]**, only one server operation (i.e. signature production, verification or archival) can be carried per request.

There are some situations, mainly in batch environments, where large numbers of operations (mainly signature creations) have to be performed. Several real examples of this need can be found in different business sectors

- Invoicing, where a company generates and signs its invoices monthly, normally in a batch process.
- Electronic Procurement, where an organization generates and signs orders and other related documents normally in batch processes.
- Email Advertising, where an organization generates and signs electronic mails in batch.
- …

In these cases, it would be very convenient to be able to carry multiple operations per request, leading to a more efficient (especially in network roundtrips) and manageable (the tracking of the *transactions* against the DSS server is less complicated) model.

This profile is concrete, can be directly implemented, and MAY be further profiled.

2.3 Relationship to Other Profiles

This profile is based directly on the **[DSSCore]**.

This profile can be combined with the Asynchronous Profile **[Async-DSS]**, especially when dealing with large requests that cannot be processed in a synchronous manner.

2.4 Signature Object

The signature object can only include signatures, therefore excluding other objects like timestamps and certificates.

2.5 Transport Binding

This profile does not constrain any transport binding defined in **[DSSCore]**.

2.6 Security Binding

This profile does not constrain any transport binding defined in **[DSSCore]**.

3 The Compound Request/Response Protocol

3.1 Element <CompoundRequest>

The `<CompoundRequest>` element is used to carry and wrap together the individual DSS-protocol requests (i.e. sign, verify or archive requests, among others that could be defined in the future) to the server for processing.

Clients can freely mix different protocol requests within the `<CompoundRequest>` element. Servers (depending of their implementation) MAY process these mixed requests, or MAY refuse them using the minor code `MixedRequestsNotSupported`.

Only one level of request-nesting is supported by this profile, therefore compound requests MUST not be carried inside other compound requests.

`<dss:OptionalInputs>` [Optional]

> Optional inputs MAY be present in order to customize how the server processes the compound request, or to carry information shared to the different requests (i.e. key information).

`<Requests>` [Required]

> Contains one or more individual requests (i.e. `<dss:SignRequest>`, `<dss:VerifyRequest>`, among others).

The `RequestID` and `Profile` attribute of the `<CompoundRequest>` element MUST be processed as described in the `<dss:RequestBaseType>` element of **[DSSCore]**.

The `RequestID` attribute for the inner requests MUST be present in every request included, in order to enable safe correlation between the individual requests and their responses.

Processing rules for the individual requests SHOULD follow the ones described for processing individual requests, except in those cases where a `<CompoundRequest>`' optional input modifies the behaviour, as described below in this document.

```
<xs:element name="CompoundRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:OptionalInputs" minOccurs="0"/>
            <xs:element name="Requests" type="dss:AnyType"/>
        </xs:sequence>
        <xs:attribute name="RequestID" type="xs:string" use="optional"/>
        <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

3.2 Element <CompoundResponse>

The `<CompoundResponse>` element is used used to carry and wrap together the individual DSS-protocol responses (i.e. sign, verify or archive responses, among others that could be defined in the future) to the client after processing.

`<Responses>` [Optional]

Contains one or more individual requests (i.e. `<dss:SignRequest>`, `<dss:VerifyRequest>`, among others). This element MUST be present when the request is successfully processed.

As the `<CompoundResponse>` is an element whose type is derived by extension from `<dss:ResponseBaseType>`, processing rules described in `<dss:ResponseBaseType>`, as described in the element of **[DSSCore]**, MUST be observed by the server when generating the responses, with the following additions

- The server doesn't need to order the inner responses in request order, as the `RequestID` of the inner requests MUST be included in all of them.
- The result of the compound response can only be Success if every inner request has also Success into its `dss:ResultMajor` element. Note that other conditions in the construction of the result of the compound response MAY apply (that is, we can have all the inner requests processed successfully and not have a Success result for the compound response).

```
<xs:element name="CompoundResponse">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="dss:ResponseBaseType">
                <xs:sequence>
                    <xs:element name="Responses" type="dss:AnyType" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```

3.3 Optional Inputs and Outputs

Apart from the optional inputs and outputs present in the inner requests and responses, the compound requests and responses MAY carry the optional inputs and outputs described below.

When considering a compound request, some of them are allowed to appear only in the compound request itself, some of them are allowed to appear only in the inner requests and some of them are allowed to appear both in the compound and the inner requests.

In order to describe these situations, this profile introduces a classification scheme for optional inputs and outputs, and provides a recommended classification under this scheme for several useful optional inputs.

The classification scheme is comprised by two categories

- **Exclusive Optional Inputs**, that is, these optional inputs that can't appear both in the compound and the inner requests for a given compound request.

- **Inclusive Optional Inputs**, that is, these optional inputs that can appear both in the compound and the inner requests for a given compound request.

  These inclusive optional inputs MAY be further qualified using a **combining algorithm** from the ones described below. The combining algorithm determines how the server processes the requests for the 2^2 possible cases

  - **None (default)**

    No combination. The optional inputs are processed independently.

    The following figure includes the four possible inputs and their outputs, where C represents the compound request, I represents a single inner request, CR represents the optional input processed by the server as the compound request's optional input and IR represents the optional input processed by the server as the inner request's optional input.

    A represents any instance of a valid optional input. Ai and Aj represent two different instances of the optional input A.

C	I	CR	IR
X	A	X	A
A	X	A	X
Ai	Aj	Ai	Aj
Aj	Ai	Aj	Ai

  - **Inherit**, that is, when the optional inputs are included in the compound request but not in the inner requests, these inner requests inherit the optional inputs from the parent one. When an optional input is present both in the compound and the inner request, this rule behaves like **None**.

C	I	CR	IR
X	A	X	A
A	X	A	A
Ai	Aj	Ai	Aj
Aj	Ai	Aj	Ai

  - **Override**, that is, every request takes the optional input from the compound request, when present.

C	I	CR	IR
X	A	X	A

A	X	A	A
Ai	Aj	Ai	Ai
Aj	Ai	Aj	Aj

- **Union**, that is, every pair of optional inputs present in the compound and the inner requests are combined using an union operation. How the union operation behaves is dependent from the optional input in question.

C	**I**	**CR**	**IR**
X	A	X	A
A	X	A	A
Ai	Aj	Ai	Ai U Aj
Aj	Ai	Aj	Aj U Ai

- **Intersection**, that is, every pair of optional inputs present in the compound and the inner requests are combined using an intersection operation. How the intersection operation behaves is dependent from the optional input in question.

C	**I**	**CR**	**IR**
X	A	X	X
A	X	A	X
Ai	Aj	Ai	Ai ∩ Aj
Aj	Ai	Aj	Aj ∩ Ai

When any combination is not supported by the server, the request MUST be refused using the minor code `NotSupported`, as defined in **[DSSCore]**.

3.3.1 Common DSS Optional Inputs and Outputs

3.3.1.1 Optional Input <dss:ServicePolicy>

As described in the `<dss:ServicePolicy>` element of **[DSSCore]**.

This optional input MUST be considered as **inclusive**, as the service policies of the compound and the inner requests MAY be different.

3.3.1.2 Optional Input <dss:ClaimedIdentity>

The presence of this optional input in a `<CompoundRequest>` is appropiate when an individual entity requests multiple operations to a server at a time, not needing to duplicate the `<dss:ClaimedIdentity>` element in every inner request, and allowing the server to perform only one authentication process that remains valid for the whole transaction.

There are other cases, i.e. when the requesters of the operations are not the clients themselves, but a proxy service acting on their behalf, where the approach described above MAY not be suitable. In these cases, the `<dss:ClaimedIdentity>` element of the `<CompoundRequest>` SHOULD not be present, in favour of the `<dss:ClaimedIdentity>` elements of the inner requests.

Therefore, this optional input MUST be considered as **exclusive**.

3.3.1.3 Optional Input <dss:Language>

As described in the `<dss:Language>` element of **[DSSCore]**.

The `<dss:Language>` element included in the compound request and the ones included in the inner requests MAY be different. This is specially useful when a proxy service is creating the compound request in behalf of several individual clients.

This optional input MUST be considered as inclusive. It’s up to the implementations to use one of the **None** or **Inherit** combining algorithms.

3.3.1.4 Optional Input <dss:AdditionalProfile>

As described in the `<dss:AdditionalProfile>` element of **[DSSCore]**.

This optional input MUST be considered as **inclusive**, as the compound and/or the inner requests MAY need different additional profiles to work properly.

This profile can be used in combination with the Asynchronous profile **[Async-DSS]**, in order to correctly process large batch-style transactions.

3.3.1.5 Optional Input <dss:Schemas>

As described in the `<dss:Schemas>` element of **[DSSCore]**.

In order to minimize duplicate schema elements (and therefore minimize the length of the request), it’s recommended to consider this optional input as **exclusive**, and to include the schemas in the compound request.

3.3.2 DSS Optional Inputs and Outputs related to signing operations

The following optional inputs and outputs SHOULD be considered when the compound request contain one or more signature creation operations. If these optional inputs are present, and no `<dss:SignRequest>` elements are present, the server MAY decide to ignore them or to refuse the request using the minor code `NotSupported`.

3.3.2.1 Optional Input <dss:KeySelector>

As described in the `<dss:KeySelector>` element of **[DSSCore]**.

The presence of this optional input in a `<CompoundRequest>` is appropiate when an individual entity requests multiple signing operations to a server at a time using the same key (i.e. a company signing invoices with its private key).

In other cases (i.e. several entities signing through a proxy service or a single entity signing with different keys), these approach MAY not be suitable, and the individual `<dss:KeySelector>` elements from the individual requests SHOULD be used instead of the one of the compound request.

Therefore, this optional input MUST be considered as **exclusive**.

3.3.2.2 Optional Input <dss:SignatureType>

As described in the `<dss:SignatureType>` element of **[DSSCore]**.

The presence of this optional input in a `<CompoundRequest>` is appropiate when an individual entity requests multiple signing operations to a server. In these cases, the desired signature type will be frequently the same.

In other cases (i.e. several entities signing through a proxy service or a single entity signing with different keys), these approach MAY not be suitable, and the individual `<dss:SignatureType>` elements from the individual requests SHOULD be used instead of the one of the compound request.

Therefore, this optional input MUST be considered as **exclusive**.

3.3.3 DSS Optional Inputs and Outputs related to verifying operations

3.3.3.1 Optional Input <dss:AdditionalKeyInfo>

As described in the `<dss:AdditionalKeyInfo>` element of **[DSSCore]**.

The presence of this optional input in a `<CompoundRequest>` may be very useful to carry information that are common to the majority or all the requests, like intermediate CA Certificates.

Additionally, there are another information whose scope is tied to a specific individual request (i.e. CRLs or OCSP responses). In these cases, the `<dss:AdditionalKeyInfo>` element of the inner request SHOULD be used.

Therefore, this optional input MUST be considered as **inclusive**, using the **Union** combining algorithm.

The union operation takes every element included in the optional input of the compound request and every element in the one included in the individual request and combines them eliminating duplicates (if any).

3.4 Result Codes

The URN used for the `<dss:ResultMajor>` elements is described in **[DSSCore]**. The URN used for the `<dss:ResultMinor>` elements MUST be `urn:oasis:names:tc:dss:1.0:profiles:compound:resultminor:` followed by the codes described below.

`<dss:ResultMajor>`	`<dss:ResultMinor>`	Description
`ResponderError`	`MixedRequestsNotSupported`	The server does not support mixed compound requests (i.e. compound requests whose inner requests are of different types)

294

4 References

4.1 Normative

[TO BE DONE]

Appendix A. Revision History

Rev	Date	By Whom	What
wd01	26/12/2005	Carlos González-Cadenas	Initial Version

Appendix B. Notices