NHS Restricted Digital Signing and Non-Repudiation						
Programme	NHS CFH	**NPFIT-FNT-TO-IG-0019**				
Sub-Prog/ Project	IT Security	*National Prog*	*Org*	*Prog /Proj*	*Doc*	*Seq*
Prog. Director	Tim Davis	**NPFIT**				
Sub Prog/ Proj Mgr	Malcolm McKeating					
Author	Andy Truscott Graham Jack John Whiteside	Version No		2.0		
Version Date	July 2007	Status		Approved		

Digital Signing and Non-Repudiation

Amendment History:

Version	Date	Amendment History
1.0	Oct 2004	Initial version, to support ETP signing requirements
1.1	Nov 2004	Minor revision to reflect changes required to align with ETP Message Signing Requirements document • General: added signing/validation "process" diagrams and restructured information to align with process • *Note that this has resulted in a change to section numbers – the numbers quoted in this table refer to the new numbering in this document* • 1: Added scope statement to limit document to content commitment signing – note intention to expand spec to cover data origin singing in future • 2/3: Added background/summary sections • 4: Added subsection to cover C14N standard – changed to reference the "exclusive" version, and provided note on impact. • 5.4.1/5.4.2: Clarified: Provided introduction clarifying usage of simple and *ni* hash forms, extracted description of transforms into introduction. • 5.4.2: Clarified handling of blank subcomponents in *ni* hash form • 5.4.2: Added process diagram to clarify *ni* hashing method • 5.3: New section providing recommendations on the implementation of C14N • 5.5: Modified to state that the signature is calculated across the entire (canonicalised) *SignedInfo* element, not just the hash • 5.6: Updated DSIG example to show encoding of a blank subcomponent within *ni* structure • 5.6: Added definition of encoding for a blank digital signature
1.2	Dec 2004	Changes to remove ni structure and sign a single hash • 5, 6: modified process diagram • 5.x, 6.x: removed references to the NI option • 5.6: replaced example
1.3	not issued	Clarifications to address supplier feedback: • 5.1: Clarified requirements for displaying information to be signed as part of a content commitment dialog • 5.5.1: Specified key search attribute values for Issuer Common Name and Policy OID. • 5.6.1: referenced a new XML schema for the XML-DSIG "profile" to be used by NPfIT • 5.6.1: fixed *SignatureMethod* reference – rsa-sha1 is the correct value • 5.6.2: clarified where/how the signing certificate is obtained
1.4	May 2005	Update to fix problems with signing standard definition and use: • 4.3: Referenced correct signing standard

		• 5: Clarified signature generation diagram • 5.5: Clarified how the PKCS#11 signature function is used to generate a signature, including options. • 6: Corrected signature validation diagram. • 6.2: Corrected description of signature validation • 6.3: Clarified status and potential form of Certificate Validation Service
1.5	October 2006	Updated references to other documents to reflect the latest versions
1.6	March 2007	Updated for IG's position regarding SHA256
1.7	June 2007	Updated Hashing requirements as per TO request.
1.8&1.9	July 2007	Updated following MM's comments on version 1.7
2.0	July 2007	Issued version of 1.9

Reviewers:

This document must be reviewed by the following. Indicate any delegation for sign off.

Name	Signature	Title / Responsibility	Date	Version
Ian Lowry		ETP Programme Manager	1.8	
Rob Gooch		ETP Technical Architect	1.8	
Malcolm McKeating		IG	1.8	

Approvals:

This document requires the following approvals.

Name	Signature	Title	Date of Issue	Version
Paul Jones		Chief Technology Officer	31/07/07	1.9
T Donohoe		Programme Director - ETP	31/07/07	1.9

Document Location

This document is only valid on the day it was printed. Please contact the Document Controller for location details or printing problems.

This is a controlled document.

On receipt of a new version, please destroy all previous versions (unless a specified earlier version is in use throughout the project).

Related Documents:

These documents will provide additional information.

Ref no	Doc Reference Number	Title	Version
1	NPFIT-NPO-GEN-IP-0067	Glossary of Terms Consolidated.doc	
2	2087 Issue 7.5	External Interface Specification	V7.5
3	NPFIT-ETP-EDB-0064	ETP Message Signing Requirements	2.0

Glossary of Terms:

List any new terms created in this document. Mail the NPO Quality Manager to have these included in the master glossary above Ref. [1].

Term	Acronym	Definition
Canonicalisation	C14N	The process of producing a physical representation, the canonical form, of an XML document that accounts for permissible changes. If two documents have the same canonical form, then the two documents are logically equivalent within the given application context.
Communications Electronic Security Group	CESG	CESG is the Information Assurance (IA) arm of GCHQ They are the UK Government's National Technical Authority for IA and amongst other matters provide advice and guidance on the usage and suitability of various cryptographic algorithyms for the public sector
Hash		A condensed representation of data – a secure hash function produces a hash value that can be considered a "fingerprint" of the data that it was computed from
Secure Hash Standard	SHA-1	FIPS standard 180-1/ 180-2 for computing a condensed representation of a message or a data file. The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest. SHA-1 produces a digest of a fixed length of 160bits
Secure Hash Standard	SHA256	FIPS standard 180-2 for computing a condensed representation of a message or a data file. The SHA-256 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very

		high probability, result in a different message digest. SHA-256 produces a digest of a fixed length of 256bits
Digital Signature		Associates the content of a message or document with a private encryption key, known only to the signer.
Signing Digital Certificate		A digital certificate that associates the identity of a signing party with a public/private key pair used by that party to sign messages. The certificate contains the public key. The associated private key is generated, stored and used on the signer's smartcard, and never leaves it.
XML-Signature		XML (Extensible Markup Language) digital signature processing rules and syntax. XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.
Certificate Authority	CA	A trusted third party responsible for issuing digital certificates as part of a registration process. In this case, the CA issues certificates on behalf of NHS CFH, for use in validating digital signatures.

Contents

1. Purpose

This document is intended to inform stakeholders as to the standards, manner, and usage of Advanced Electronic Signatures (Digital Signatures) within NCRS compliant systems from Release 2006B onwards.

Two uses for digital signatures are considered:

- **Content Commitment Signatures** – where the signature is used for specific messages to affirm the signer's commitment to selected content in the message – the signer is prompted to confirm that the content should be signed
- **Origin Authentication Signatures** – where the signature is included automatically on all messages sent by a user or system, to authenticate the identity of the sender

In this version of the document, only content commitment signatures are addressed. A future version of the document will be expanded to address origin authentication signatures as and when their use is required.

2. Background

Content commitment signatures are implemented for applications within the NHS CFH programme where an "advanced electronic signature" is required as equivalent of a hand written signature in indelible ink (such as ETP in Release 1).

This document focuses on requirements that apply across all application domains that utilise content commitment signatures, and assumes that domain-specific issues – such as the messages which require a signature and the XML elements to be covered by the signature – are defined by related documents owned by the application domain itself.

3. Summary

This document:

- Summarises the standards to be used in the generation and validation of Advanced Electronic Signatures
- Provides an overview of the processes to be implemented for signature generate and validation
- Defines the requirements that a compliant application must satisfy in generating and validating Advanced Electronic Signatures

As noted above, the authority will define domain-specific requirements for each application that requires Advanced Electronic Signatures.

4. Summary of the Standards to be used

4.1. XML-Signature (XML-DSIG)

XML-DSIG is an XML compliant syntax which can be used for the signing of message parts. It is defined at http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

4.2. XML Canonicalisation

Canonicalisation (C14N) is the process of producing a physical representation, the canonical form, of an XML document that accounts for permissible changes. If two documents have the same canonical form, then the two documents are logically equivalent within the given application context. Since the XML context in which the signed fragments are contained may change it is necessary to use a canonicalisation algorithm that takes account of this.

The standard to be used is Exclusive Canonical XML (without comments), as defined at http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/.

> ***Note*** *that currently-available XML tooling may not implement this more-recent W3C XML C14N recommendation. If that is the case, then it will be necessary for an application to post-process the canonicalised XML to implement the additional requirement (as described in section 3 of the referenced recommendation: essentially that unused namespaces are not declared, and elements in the xml: namespace are not inserted by the canonicalisation process).*
> *Additionally, for interoperability it is a requirement that systems remove any whitespace outside of elements during the canonicalisation process. This is described in more depth later in this document.*

4.3. PKCS#1 (RSA-SHA1)

PKCS#1 (originally http://www.ietf.org/rfc/rfc2437.txt now superseded by http://www.ietf.org/rfc/rfc3447.txt) contains a range of recommendations relating to the use of the RSA algorithm to implement public key cryptography, including signature applications.

In this document the XML-Signature standard is specified to be used with SignatureMethod of *RSA-SHA1*, which equates to the use of the RSASSA-PKCS1-v1_5 algorithm as defined in PKCS#1 (Section 8.1 in the referenced RFC) in combination with the SHA1 algorithm referenced below.

4.4. SHA1 (160bit)

The SHA1 (Secure Hash Standard) is an algorithm that can be used to create a representation of the information to be signed. The standard can be found exhaustively described at http://www.itl.nist.gov/fipspubs/fip180-1.htm and http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

4.5. PKCS#1 (RSA-SHA256)

PKCS#1 (http://www.ietf.org/rfc/rfc3447.txt) contains a range of recommendations relating to the use of the RSA algorithm to implement public key cryptography, including signature applications.

In this document the XML-Signature standard is specified to be used with SignatureMethod of *RSA-SHA256*, which equates to the use of the RSASSA-PKCS1-v1_5 algorithm as defined in PKCS#1 (Section 8.2 in the referenced RFC) in combination with the SHA256 algorithm referenced below.

4.6. SHA256 (256bit)

The SHA256 (Secure Hash Standard) is an algorithm that can be used to create a representation of the information to be signed. The standard can be found exhaustively described at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

4.7. PKCS#11

This is a standard 'C' API known as "`cryptoki`" that provides access to the cryptographic capabilities of a device (known as a cryptographic "token"). The PKCS#11 standard is defined by RSA Labs. Full details can be found at:

ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/201final/spec/v201.pdf

The API provides an interface to the functions, keys and certificates held on the smartcard of an NCRS user, and is used by compliant applications to support the generation of signatures.

An implementation of version 2.01 of this API is provided as part of the SPINE Security Broker – Identity Agent component.

In addition a Java based version of the interface is provided via a wrapper from IAIK.

A subset of the API's functions and associated signing mechanisms, appropriate for the operations to be supported by compliant systems, is documented in [5]: *SPINE External Interface Specification V7.5*[1].

4.8. The use of SHA-1 and SHA-256 algorithms for signing

The American National Institute of Standards and Technology (NIST) have stated that although SHA-1 is not yet broken, advances in computer processing cabability make it prudent to phase out SHA-1 for digital

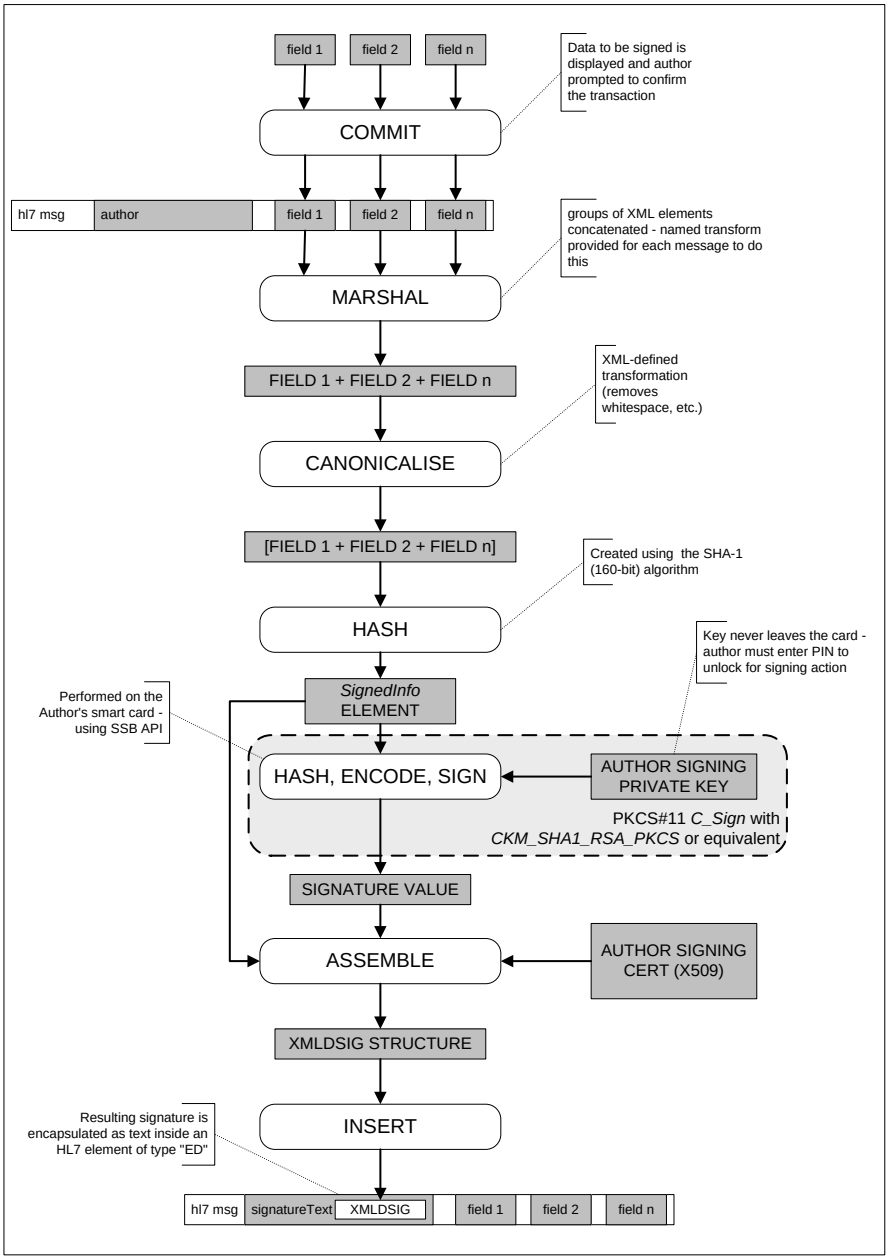
[1] The current version specifies the functions but not the signing *mechanisms* subset, a future version of the specification is expected to do this.

signatures and digital time stamping by 2010. The Communications Electronic Security Group (the information assurance arm of the Government Communications Headquarters (GCHQ)) support this stance.

Plans are currently being put in place to migrate CFH from SHA-1 to SHA-256 for use in advanced electronic signatures.

5. Signature Generation

The process of producing a signature for a message is summarised in the attached diagram:



5.1. Content Commitment (Non Repudiation)

For each message that requires a content commitment signature, a compliant system must implement a content commitment dialogue with the signing user, as defined here.

Before signing a message the system must prompt the user to affirm that they wish to proceed. The content commitment display and dialogue must comply with the following constraints:

- The system must display a screen reflecting the information that will be included within the signature
    - Note that it is not required that a compliant system displays the actual XML of the fields to be signed, or even all of the fields that will be included (for example, it may not be useful to display codes such as OIDs and message IDs that are not meaningful to users).
    - The precise set of fields that must be included within a signature and those that must be displayed as part of the content commitment dialog are defined for each message by the relevant application domain.
- The prompt displayed to the signer must include text, to be defined by the authority, that makes the commitment clear,
- The prompt must provide clear options that allow the signer to proceed or cancel.

If required, a compliant system may implement a batch signing process to streamline workflow. In this case, the overall content commitment dialogue must meet the above requirements.

Immediately before a signature is applied to message, a compliant system must confirm that the current user of the system is the owner of the card that will be used. To do this the system must prompt the user for entry of a PIN, and pass the entered PIN to the card for authentication."

In practical terms, this means that the system must implement a login/logout pair of calls around the signing operation. The system must

- Use a freshly entered PIN rather than one entered earlier in the session,
- Immediately after use, overwrite memory locations that it used to store or process the PIN,
- ensure that the PIN is not included in trace files or audit logs that it generates.

5.2. Marshal

Marshalling involves gathering the data that is going to be included in the signature.

An application that implements advanced electronic signatures must define the data elements that will be covered by the signature for each message that requires signing.

Apart from the case where the signature is to cover the entire message as sent, an application must provide an *XSLT* transform for each message. The message can be run through the transform to automatically marshal the data to be signed into a separate XML document.

Alternatively the data can be marshalled "manually", using the named transform as a definition of the data elements to be included.

Note that the entire document output from the transform, including the "fragment" tags, is fed into the subsequent c14n hashing steps.

5.3. Canonicalisation (C14N)

The data to be hashed must be canonicalised to ensure that the hashes calculated by the signer and verifier match, independent of the way that the XML is encoded.

The referenced standard XML canonicalisation method must be used – note that the exclusive, "without comments" version of the standard is mandated.

In addition to the C14N canonicalisation, to ensure interoperability it is a requirement that any whitespace outside of any elements is removed (If the supplied transform was used for the marshalling steps above this should already have taken place). Ensuring that this whitespace is removed before hashing the stream is a requirement to ensure consistent hashes are created (and can be validated) across supplier systems.

As a diagnostic precaution, it is recommended that suppliers generating and validating advanced electronic signatures implement a trace function which can capture the extracted fragment and its canonical form for use in resolving validation issues.

It is noted that as part of the development of the XML C14N recommendation, interoperability testing was performed using a test message suite. In the event of an issue arising, it may appropriate to use this suite as the basis for comparing implementations.

5.4. Hashing

Hashing is performed using either the SHA1 or SHA256 algorithm referenced above. Please see domain specific documentation for which algorithm is to be used.

The W3C canonical form of the XML to be signed is hashed, and the resulting value is inserted directly in the XML DSIG *digestValue* element as discussed below along with the relevant algorithm identifier (uri).

5.5. Signing

Signature Generation is performed according to the requirements defined in the XML DSIG recommendation.

Signing is performed on the entire *SignedInfo* element. Before signing, the *SignedInfo* element must be canonicalised using the referenced C14N method. At this point it is a requirement that any whitespace outside of any elements is removed (as in the canonicalisation step above). A transform has been included with the document "Message Signing Requirements" – NPFIT-ETP-EDB-0064 that will remove any superfluous whitespace within the signedInfo Element. Please note that this transform has not been tested with all xslt engines. It is not mandated that suppliers use this transform as a method of removing the superfluous whitespace but it is a requirement that suppliers remove this whitespace from the canonical form before hashing is performed (by whatever method they choose).

The signature value is generated using the RSASSA-PKCS1-v1_5 algorithm referenced above. The advanced electronic signature is generated by a signing function on the smartcard of the person signing, using a private signature key (also know as a Content Commitment Credential) stored on the card.

The Spine Security Broker Identity Agent component, provided as part of the 2006B2 SPINE release, provides a Content Commitment API to invoke this function. Both 'C' and Java APIs are provided. See section 4.7, PKCS#11 for more information.

The PKCS#11 signing function, *C_Sign* can be used in two ways, depending on the needs of the application and local design considerations:

- The function accepts the data to be signed (i.e. the *SignedInfo* element) and returns the signature value to the application as an octet stream. This is the *CKM_SHA1_RSA_PKCS*[2] mechanism used in conjunction with *C_Sign*.
- The application hashes the *SignedInfo* element and encodes as specified by the RSASSA-PKCS1-v1_5 algorithm (and summarised in the XML-Signature standard), this value is then passed to the function which returns the signature value as an octet stream. This is the *CKM_RSA_PKCS* mechanism used in conjunction with *C_Sign*.

See [2] and the referenced PKCS#11 standard for more information.

The returned signature value is stored as a base64-encoded value in the *SignatureValue* element of the DSIG XML structure described below.

[2] At the time of writing the current version of the Identify Agent is based upon Version 2.01 of PKCS#11. This version does not include the relevant mechanism for SHA256 (CKM_SHA256_RSA_PKCS) This mechanism is available in V2.20 of PKCS#11. Until the Identity Agent is upgraded to version2.2, it is suggested that suppliers progress using the second method available described above.

The signing application must “login” to the card to unlock its capabilities before calling the signing function, and “logout” afterwards. The login functions accepts the PIN entered as part of the content commitment dialogue, and returns an error if the PIN is not correct.

A system may implement a batch signing function by calling the signing function multiple times between login/logout calls.

5.5.1. Signing Key Selection

A user’s smartcard contains multiple private keys, along with their associated public key digital certificates. To perform the signing operation, the application must first select the key/certificate that has been allocated for signing; for ETP this will be a Content Commitment certificate with the key usage bit asserted for non repudiation only. For Authentication of data origin, the key usage bit will be asserted for authentication only. Signing certificates will not have any other bits asserted in the key usage field.

The Content Commitment API provides functions to query the card and filter certificates, using a set of specified attributes. The result is a PKCS#11 object reference for the matching private key object. This object reference is passed to the signing function, for use in computing the signature value.

The set of attributes required for the search are defined by the Content Commitment API in [2] and include constants, such as the Issuing Authority for the associated digital certificate, and variables, specifically the Distinguished Name of the person doing the signing.

The following attribute values must be used to identify the correct key/certificate in all CFH sandpit and live environments:

Item	***NIS Environment***	***Live Environment***
	SHA1	***SHA1***
Issuer Common Name	NISX_SubCAcc	nhs level 1b
Policy OID	1.2.826.0.1275.102.0.3	1.2.826.0.1275.102.0.3
Signature Algorithm Name	SHA1withRSA	SHA1withRSA
Signature Algorithm OID	1.2.840.113549.1.1.5	1.2.840.113549.1.1.5
Key Usage OID Check - for NonRepudiation (i.e. true) (x509 V3 only)	2.5.29.15	2.5.29.15

Note on the Policy OID

The policy OID is an NHS OID which is formed from the following constituent parts;

1.2.826.0.1275.**VVV.W.X.Y.Z**

1.2.826.0.1275 represents the NHS UK OID

VVV is the PKI OID (101= Authentication, 102=Content Commitment)

W is the Country ID (0 = England)

X is the authentication level (i.e. 3 = e-GIF level 3)

Y is the major revision level

Z is the minor revision level

When verifying the certificate to be used for content commitment, the OID upto but not including the revision levels should be used. I.e. check the left part of the policy OID = 1.2.826.0.1275.102.0.3

Note on the keyUsage return values

The key usage values are returned from the X509 certificate as an array of Booleans. The values and positions in the array are as follows;

(0) digitalSignature

(1) nonRepudiation

(2) keyEncipherment

(3) dataEncipherment

(4) keyAgreement

(5) keyCertSign

(6) crlSign

(7) encipherOnly

(8) decipherOnly

For use in signing, the prescribing systems aim is to validate that the 2nd element in the array (1 – nonRepudiation) is set to TRUE

5.6. Assemble

5.6.1. The DSIG Structure

The signature value and its supporting hash is carried in an XML DSIG structure as referenced above.

The following examples show how this structure is used and defines the required values for the various attributes and references. An XML schema which reflects this is provided in the MiM for use in validating generated and received signature structures.

Example Based on SHA1 hashing[3]

```
<?xml version="1.0" encoding="UTF-8"?>
<signatureText xmlns="urn:hl7-org:v3" mediaType="application/xml">
    <!-- the mediaType is fixed to application/xml -->
    <!—note: the signatureText element is from the HL7 message that includes the signature -->
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <!-- The method that is used to produce the canonicalized form of the data within the
            SignedInfo element -->
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <!-- The method used for generating the signature (of the contents of SignedInfo) -->
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <!-- The object that is being digitally signed -->
            <Reference>
                <!-- The transforms that have been applied to the object that was digested -->
                <Transforms>
                    <!-- The input to the digest is canonicalised -->
                    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <!-- The encoded value of the digest. "A SHA-1 digest is a 160-bit string. The
                content of the DigestValue element shall be the base64 encoding of this bit string
                viewed as a 20-octet octet stream" from http://www.w3.org/TR/xmldsig-core -->
                <DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
            </Reference>
        </SignedInfo>
        <!-- The value of the advanced electronic signature -->
        <SignatureValue>MCwCFEJKOWj3Iii/7Aq+WZfhE3+SfiUOAhRSXEjd+fbcc/HlS/xSphuHG
        NKqRw==</SignatureValue>
        <!-- The key needed to validate the signature -->
        <KeyInfo>
            <X509Data>
                <!--Signer cert base64 encoded as described in http://www.w3.org/TR/2002/REC-
                xmldsig-core-20020212/Overview.html#sec-X509Data -->
                <X509Certificate>/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZ
                PY1Y+r/F9bow9subVWzXmZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYV
                DwT7g/bTxR7DAjVUE1oWdfOuK2HXKu/yIgMZndFIAcc=</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</signatureText>
```

[3] SignatureMethod Algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1 taken from http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (rfc3275)

DigestMethod Algorithm http://www.w3.org/2000/09/xmldsig#sha1 taken from http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (rfc3275)

Example based on SHA256 hashing[4]

```
<?xml version="1.0" encoding="UTF-8"?>
<signatureText xmlns="urn:hl7-org:v3" mediaType="application/xml">
    <!-- the mediaType is fixed to application/xml -->
    <!—note: the signatureText element is from the HL7 message that includes the signature -->
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <!-- The method that is used to produce the canonicalized form of the data within the
            SignedInfo element -->
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <!-- The method used for generating the signature (of the contents of SignedInfo) -->
            <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256"/>
            <!-- The object that is being digitally signed -->
            <Reference>
                <!-- The transforms that have been applied to the object that was digested -->
                <Transforms>
                    <!-- The input to the digest is canonicalised -->
                    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <!-- The encoded value of the digest. "A SHA-256 digest is a 256-bit string. The
                content of the DigestValue element shall be the base64 encoding of this bit string
                viewed as a 32-octet octet stream" -->
                <DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
            </Reference>
        </SignedInfo>
        <!-- The value of the advanced electronic signature -->
        <SignatureValue>MCwCFEJKOWj3Iii/7Aq+WZfhE3+SfiUOAhRSXEjd+fbcc/HlS/xSphuHG
        NKqRw==</SignatureValue>
        <!-- The key needed to validate the signature -->
        <KeyInfo>
            <X509Data>
                <!--Signer cert base64 encoded as described in http://www.w3.org/TR/2002/REC-
                xmldsig-core-20020212/Overview.html#sec-X509Data -->
                <X509Certificate>/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZ
                PY1Y+r/F9bow9subVWzXmZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYV
                DwT7g/bTxR7DAjVUE1oWdfOuK2HXKu/yIgMZndFIAcc=</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</signatureText>
```

[4] SignatureMethod Algorithm http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 taken from http://www.ietf.org/rfc/rfc4051.txt

DigestMethod Algorithm http://www.w3.org/2001/04/xmlenc#sha256 taken from http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

5.6.2. X509 Certificate

The signer's content commitment digital certificate (which is used to validate the signature) must be transported along with the signature.

It is inserted into the DSIG XML structure, in the *X509Certificate* element as a base64 encoding of the certificate octet stream.

The Content Commitment API provides functions which can be used to retrieve this certificate from the smartcard. The relevant key to retrieve is identified as part of the search discussed above in section 5.5.1, Signing Key Selection.

5.7. Insert

The resulting XML DSIG structure is transported as part of the message with which it is associated – each message which requires an advanced electronic signature defines a nominated data element to carry it.

The signature structure is inserted as text into the nominated data element, which must be of type *ED* as defined by HL7. The attributes of the data element must be set as follows:

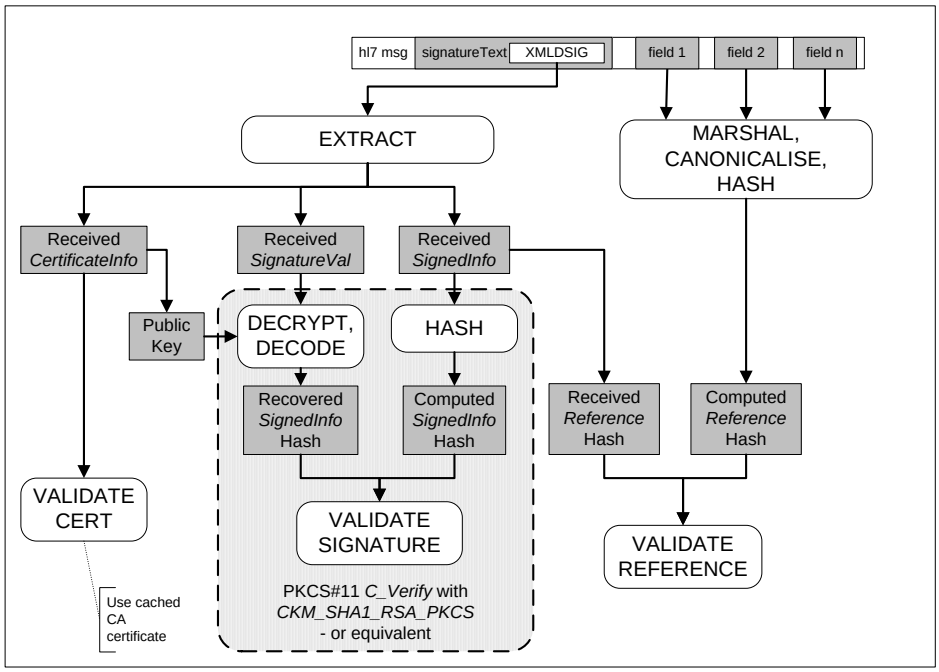
```
<?xml version="1.0" encoding="UTF-8"?>
<signatureText xmlns="urn:hl7-org:v3" mediaType="application/xml">
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
.
.
.
.
        </KeyInfo>
    </Signature>
</signatureText>
```

In the case where a message definition includes a *signatureText* element but no signature is required, the element is coded as:

```
    <signatureText nullFlavor="NA"/>
```

6. Signature Validation

The process for validating a signature is summarised in the following diagram:



Validation of a received signature consists of three steps:

- Validate Reference – which checks that the information received matches the hash information received,
- Validate Signature – which confirms that the hash information received is that which was signed, and
- Validate Certificate – which confirms the identity of the party who created the signature.

A failure in any one of these steps indicates a signature validation failure. Requirements for application behaviour in the event of a validation failure are defined by the application domain in which the signature is implemented.

The requirements for hash comparison and signature validation are defined in the following subsections.

The requirements for certificate validation are defined by the application domain in which the signature is implemented – available options for certificate validation are listed below.

6.1. Validate Reference

- The hash value for the received message is computed in the same way as defined for generating a signature above. The algorithm to be used in re-creating this hash value is identified by the uri detailed in the algorithm attribute of the DigestMethod element.

- The resulting hash value is then compared with that received in the *DigestValue* element.
- A failed match results in a signature validation failure.

6.2. Validate Signature

Signature validation is performed according to the requirements of the XML DSIG recommendation, and the associated PKCS#1 RSASSA signature method. In summary:

- The signer's public key is extracted from the received X.509 certificate in the *KeyInfo* element,
- This key is used to decrypt the received *SignatureValue*, recovering the hash of the *SignedInfo* block that was signed,
- The received *SignedInfo* block is canonicalised and hashed as per the signing rules (again, the algorithm to be used is identified by the uri within the algorithm attribute of the digestMethod element),
- The recovered hash is compared with the hash of the received SignedInfo block,
- A failed compare results in a signature validation failure.

The NCRS does not provide API support for signature validation. For information, PKCS#11 includes a *C_Sign* function that can be used in conjunction with mechanism *CKM_SHA1_RSA_PKCS, CKM_SHA256_RSA_PKCS*[5] or *CKM_RSA_PKCS* to assist with validation.

6.3. Validate Certificate

Methods to validate the certificate include:

- Validity date – was the signature generated during the period for which the certificate was valid? [6]
- Certificate chain – is there a chain of valid CA certificates that link this certificate to a trusted root certificate?
- Certificate signature – is the signature on this certificate valid?
- Revocation status – has this certificate been revoked? [7,8,9]

[5] At the time of writing the current version of the Identify Agent is based upon Version 2.01 of PKCS#11. This version does not include the mechanism CKM_SHA256_RSA_PKCS This mechanism is available in V2.20 of PKCS#11. Until the Identity Agent is upgraded to version2.2, it is suggested that suppliers progress using CKM_RSA_PKCS mechanism as described in the signature creation section above.

[6] Note that the answer to this question depends upon a trusted time value for the signature, which may not be available.

The authority's requirements for a particular application domain define which of these options are to be implemented for signature validation in any given case (for ETP see reference [3] Message Signing Requirements).

[7] Like "validity date", this question requires a signature time value that can be trusted.

[8] This check requires the support of a certificate validation service, which has not been implemented at this stage. It is currently assumed that this service may be provided as part of the NCRS SPINE, in which case a future version of the External Interface Specification [6] will define the interface and usage of this service.

[9] It is currently assumed that this service will be an implementation of the X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, as defined in RFC 2560 from the IETF. The service will be available over HTTP and will accept a certificate identifier and return a validity indication – either "good", or "revoked" in which case a date/time and reason will be returned.