Chunk Registration Request

for Enabling the PNG Standard

to Support Digital Signatures

Specification, Technical Proposal & Request

Thomas Kopp / Dialogika GmbH

Document History		
Version	**Date**	**Remarks**
1.0	18.02.2008	Initial Version
1.1	20.02.2008	Minor Syntactical Corrections
1.2	19.03.2008	Renamed Chunk, Added Introductory Chunk & Re-signing Support
1.3	23.03.2008	Syntactical Corrections and Amendments (Glenn Randers-Pehrson)

Table of Contents

1 DISCLAIMER

This document proposes a new PNG chunk type and **requests its registration** in order to support PNG digital signatures in a generic and extensible manner.

Neither Dialogika GmbH nor LuxTrust S.A. claim any intellectual property rights or patents or protection of other rights concerning this proposal by virtue of authoring it.

In any event, if the W3C standardization group accepts registering the chunk type specified below, we would like the Internet community to benefit from the new feature and use it in an open and interoperable way so as to make networks and communications more secure.

LuxTrust S.A.

Based in Luxembourg, LuxTrust S.A. was founded in November 2005 by the Luxembourg Government and a number of key corporate players in the public and private sector in Luxembourg so as to cater to the increasing need for security and confidentiality in electronic commerce and in Internet-based relationships between citizens, government authorities and companies.

LuxTrust S.A. is a certification authority supplying electronic certificates for the authentication of individuals and providing secure electronic signatures in Internet- and intranet-based transactions. As a trusted third party, LuxTrust also guarantees high-security electronic certification services.

For more information about LuxTrust please visit *www.luxtrust.lu*, *Qui sommes nous?*.

DIaLOGIKa GmbH

Founded in 1982, DIaLOGIKa is a German systems and software house that conducts projects on behalf of industry, finance, and governmental and supranational clients such as the institutions of the European Union (EU).

DIaLOGIKa has contributed to the open source community by actively participating in the development of the W3C's http reference server. DIaLOGIKa has adopted Java right from the very beginning for creating software and solutions, in addition to actively promoting the further development of Java by joining the JCP (Java Community Process) in 2001.

Since its founding DIaLOGIKa has focused on technically demanding projects in the field of multilingual text and data processing, security solutions, PKI consulting, specification and implementation.

2 ACKNOWLEDGEMENT

Various helpful criticisms and improvements of the initial proposal have been contributed by various members of the W3C and SourceForge PNG groups. Thus, we would like to extend a special word of thanks to

- *Chris Lilley* for managing discussion lists and activities

- *Glenn Randers-Pehrson* for supplying essential improvements in order to provide one-pass verification and re-signing features due to an introductory chunk and signature compatibility with the MNG format

- *Greg Roelofs* for proofreading and verifying the proposal

- *Willem van Schaik* for critical suggestions concerning naming and structuring

Thanks also to people at LuxTrust and DIaLOGIKa for contributing to the initial proposal and ongoing maintenance of the specification

- *Fréderic Foeteler* for proofreading and verifying the proposal

- *Guy Muller,* for discussing security and compatibility issues

- *Marcus Schmidt* for proofreading the proposal

- *Martin Boßlet* for supplying helpful suggestions concerning compatibility with the existing PNG specification and concerning discussions about potential security issues as well as supplying a proof of concept in developing a prototype PNG encoder/decoder with signature generation/verification

- *Remy Els* for proofreading and discussions about the structural approach

Special thanks to

- *Amy Bryant* for revising and proofreading the document in order to put it into comprehensive and correct English

Finally thanks to all those who have directly or indirectly contributed to this specification and who I might have forgotten to mention explicitly; and also to those who might contribute to its maintenance in the future.

3 PROBLEM STATEMENT

Portable Networks Graphics (PNG) is a powerful ISO/W3C standard for digitally storing images. PNG is patent-free and has thus become very popular. It is also capable of replacing other formats like GIF, JPEG or TIFF. Finally, PNG is extensible by adding so-called **extension chunks**, with well-typed chunks being the atomic storage unit of PNG.

The more PNG is used for storing scanned documents, the more digital signatures will become an important common requirement for PNG. Unfortunately, the *PNG standard does not currently address digital signatures.*

4 SOLUTION OPTIONS

We discuss three possible scenarios for extending the PNG format with digital signature support:

4.1 Enveloping Signature

This approach attaches a digital signature by encapsulating the original document (image) in an envelope, with the envelope storing signature-related information, i.e. the signature itself and other items like signer certificates and timestamp signatures.

An adopted international standard exists, namely Cryptographic Message Syntax (CMS), for implementing such a format (cf. RFC 3852 and RFC 4853 for details). CMS supports envelopes for various purposes, with the **signed-data** content type (and optionally the **authenticated-data** content type) being used for the signature aspect.

Note that in this specification proposal, the word "CMS" always means Cryptographic Message Syntax, not Color Management System.

4.1.1 Advantages

CMS is a well-defined **standard**, providing a generic format for storing digital signing data of any kind. Consequently, this approach could also be applied to a PNG data stream.

Existing CMS software like parsers, generators etc. could deal with this format and hence be able to understand and verify signed PNG data streams.

Since the format would rely on an adopted standard, it would be **robust and perennial** and also completely independent of future updates of the PNG format. In addition, the CMS format itself is designed to be extensible concerning future cryptographic algorithms without its syntax having to be modified.

Since the CMS and the PNG format are designed to support **streamed processing**, signature generation and verification does not require exhaustive processing and buffering capabilities. Consequently, even **huge images** could be processed on limited devices, e.g. handhelds.

4.1.2 Disadvantages

Using a container structure like an enveloping CMS signed-data format makes the original format **unreadable** for existing PNG processing tools (editors and/or viewers). Consequently, special handling would be required for making the signed content readable again, e.g. by additionally providing some kind of PNG wrapping/unwrapping tools.

However, processing PNG data streams in this way is not useful at all. In particular, it does not permit a digitally signed PNG to be published together with its signature on a website for processing by ordinary web browsers. Thus, an enveloping PNG digiatal signature would have a substantial **impact with regard to interoperability**.

Hence, the enveloping signature approach would only be applicable to environments not requiring a high level of interoperability.

4.2 Detached Signature

This approach stores digital signatures separately of the signed data stream.

4.2.1 Advantages

This approach does not require any special format support for signing PNGs. A reasonable specification of the PNG-specific signing and verification processing would suffice to support digitally signed PNGs.

PNG digital signature information could be stored using any format, e.g. the CMS standard, which also provides for storing signed data objects separately of their signature information.

Existing tools and environments would not be impacted.

4.2.2 Disadvantages

Storing signed data separately of its signature information generally causes logistic problems over time when dealing with large amounts of data.

It would be additionally necessary to always link both elements involved and to set up a suitable infrastructure for signing, storing, exchanging and verifying PNG data streams.

Setting up such an infrastructure would increase processing overhead and also result in a major impact on overall PNG handling when dealing with signatures.

4.3 Enveloped Signature

This approach embeds a digital signature in the original document (image). In order to do this, the document (image) format in question has to provide a storage structure (sub-element) for digital signature information.

Unfortunately PNG has no defined standard for embedding digital signature information. Nevertheless, PNG is extendable and offers a simple way of adding a dedicated extension chunk in order to embed a digital signature in a PNG data stream (cf. sections below for technical details).

4.3.1 Advantages

Using the original document (image) format with an embedded signature provides compatibility with existing tools. In the case of PNG, decoders or viewers would discard a new, hence unknown extension of this type and treat an attached signature as if it weren't there. (cf. *error checking part* of the PNG standard: An unknown chunk type is **not** to be treated as an error unless it is a critical chunk.) As a consequence, digitally signed **PNGs** could **envelope their own signature** and at the same time be processed and/or viewed without impacting existing tools.

Example: the above-cited use case — which stores signed images on a website for optionally checking their signature — would be possible and still **interoperate with** existing **browsers**. By contrast, software aware of such an extension could check the stored information and make the verification result transparent to the user, e.g. a special browser filter could filter a web page and exclude those images that are not properly signed, e.g. for avoiding images containing malicious script includes.

Note that a significant feature of this approach, i.e. using PNG extensions, is that the viewable **content** of the image would **not** be **modified** when attaching a signature and consequently be **fully preserved in its original form**.

Other image signing approaches exist which integrate digital signatures in viewable content, e.g. by using unused alpha channels or other superposition methods. These approaches effectively modify the original content. Although the modification may not be visible, these approaches would cause subtle changes of the original document (image), thus potentially impacting legal validity.

4.3.2 Disadvantages

Special provisions are required for additionally supporting re-signing of signed images and for supporting streaming capability with regard to signature verification.

Embedded signatures **rely on** their **container format**. Consequently, software developed for signing and verifying signatures is format-specific and cannot be re-used in a generic way.

5 SOLUTION PROPOSAL

After having pondered the implications of the above-discussed options, we **propose** using the **enveloped signature** approach for PNG. An important feature of this alternative is the sizable advantage of being interoperable with the large set of existing PNG viewers and editors.

In addition, we suggest using CMS (cf. RFC 3852 and RFC 4853 for details) as the embedded signature information container format. This method provides additional advantages:

- As already mentioned, CMS is a well-known, widely adopted international standard and robust against format changes even when new cryptographic algorithms are used in the future.
- Using an adopted standard for storing sensitive information avoids the possible risk of potential security issues which might be caused by using a proprietary storage format.
- This approach would still opt in favor of using enveloping, detached and enveloped signatures in any acceptable combination.

6 REGISTRATION REQUEST

In order to support enveloped digital signatures for the PNG standard, we request that the new PNG chunk type specified below be registered as an optional and non-visible part of the data format for storing signature information in a generic manner.

6.1 Format Specification

This proposal specifies support for **PNG digital signatures** in order to protect them against tampering, e.g. when storing PNGs on a website or in an archive. It also opts for supporting other features, e.g. **message authentication codes** used for protecting PNG data streams during transmission between two or more parties. Although such options are addressed, the major focus is directed to digital signatures only, which conformant generators/verifiers are required to support.

Note that tampering protection concerns an entire PNG data stream. There is a subtle aspect that PNG image chunks may be **re-ordered** under certain conditions and still result in the same visual representation. However, the tampering protection support proposed here would result in a signature being breached by re-ordering chunks in any way. This **strict approach** has been chosen due to the fact that the PNG standard does not define any equivalence relationship for chunk re-ordering of a given PNG and consequently lacks a canonical format that could be the basis for a lenient approach.

Nevertheless, this proposal would still be compatible with a lenient approach, provided that a canonical PNG data stream format were to be defined in the future (cf. data format section below for details concerning this option.)

Also note that this specification proposal supports **multiple signers in parallel** and/or **re-signing** a signed PNG with optionally preserving existing signatures.

This proposal also opts for signing the entire content only in order to transparently support the "you should only sign what you see" paradigm.

The proposal additionally addresses **preserving** digital signatures **when embedding** a PNG in an MNG.

Furthermore, streaming capability is supported, which addresses the issue of processing **huge images** on devices with limited resources.

Note that an important feature of the enveloped approach is its compatibility with existing editors and viewers, so that a signed PNG does not require any special processing for being visualized.

We propose the format (extension chunk) specified below, which is in accordance with the existing PNG standard and compatible with existing processing tools. (*Note that the sections in italics are for explanatory purposes only and do not specify additional rules.*)

6.1.1 Chunk Type

The new chunk type should be **ancillary**, as indicated by the lower-case beginning of its name.

Although the chunk content concerns the entire PNG data stream, this requirement exists due to the fact that digitally signing a PNG data stream should not be mandatory. Additionally, it provides compatibility with existing processing tools, i.e. editors and viewers, which should bypass optional chunks they don't understand.

6.1.2 Text Keyword

The new optional PNG extension chunk should be named **dSIG** ("Digital *Signature*"), with the chunk type's precise hexadecimal coding being **0x64 0x53 0x49 0x47.**

*The chunk should be used for **optionally** storing digital signature information concerning the entire PNG data stream.*

6.1.3 Ordering and Multiplicity

The new dSIG chunk type must always be **arranged in pairs**, i.e. an **introductory dSIG** chunk indicates the signature algorithm used for signing the data; a corresponding

terminating dSIG chunk stores the signature including accompanying information, e.g. the signer certificates.

A dSIG chunk pair may properly nest other dSIG chunk pairs in order to **re-sign a signed PNG**, with the introductory chunk of the outer pair immediately preceding the introductory chunk of the inner pair without any different chunk type in-between and with the terminating chunk of the outer pair immediately following the terminating chunk of the inner pair without any different chunk type in-between.

The introductory chunk of the outermost dSIG pair must directly follow the image header (IHDR). The terminating chunk of the outermost dSIG pair must directly precede the image trailer (IEND).

Note that a dSIG chunk pair always stores a signature based on a **message digest** calculated over the image header, the **entire data stream** between the current dSIG chunk pair, and the image trailer in exactly this order. Each octet of a chunk used for digest calculation must be taken into account for adhering or verifying a signature, i.e. including all length, type, data and CRC bytes. Chunks have to be used for digest calculation in exactly the same order as they appear in the original PNG data stream. This would accordingly apply to **message authentication codes**.

Also note that the **8-byte PNG signature** is not included in any digest calculation. By contrast, when verifying a digital signature of a standalone PNG it must always be verified that the **image header immediately follows a valid 8-byte PNG signature** without any octets in-between. Signature verification of a standalone PNG **must fail** if this is not the case.

Note that the special treatment of the 8-byte PNG signature concerning the digest calculation opts for preserving a signature when embedding a signed standalone PNG into an MNG. A digital signature of a PNG embedded in an MNG protects the embedded structure only. However, the digital signature of a standalone PNG protects the entire PNG data stream against tampering due to the requirement specified in the previous paragraph. Also note that the special check concerning the 8-byte PNG signature of a standalone PNG is ***crucial for security****.*

"(8-byte) PNG signature" as used in the previous two paragraphs possesses the semantics specified in the PNG standard, not the semantics of a digital signature.

The **ordering** *requirements specified opt for calculating a PNG digest with a minimum buffering overhead. Consequently, this proposal supports a streaming capability of signature processing (signing and signature verification), which is a key aspect with regard to the processing performance and handling of sizable amounts of image data.*

Multiple dSIG chunk pairs allow for repeatedly signing a PNG data stream, with multiple signing processes being reflected in the individual dSIG chunk pairs.

Introductory dSIG chunks can be distinguished from terminating dSIG chunks on account of different chunks in-between. Note that a PNG has at least one IDAT chunk. Corresponding chunks of a dedicated dSIG chunk pair can be identified due to the proper nesting rule.

6.1.4 Data Format

The overall format of a **dSIG** chunk should be exactly as specified in the PNG standard, i.e. comprising the type, length, data and checksum parts.

The data part of a **dSIG** chunk should be a **signed-data content type** according to the Cryptographic Message Syntax Specification (CMS) (cf. RFC 3852 and RFC 4853 for details), the above-cited container type being described in detail in section 5 of RFC 3852.

Note that this format is used for compatibility with adopted signature standards and hence compatibility with existing implementations for signing data and also compatibility with future evolutions of these standards, which are designed to be extensible without being breached by new algorithms.

Additionally, this format avoids possible security issues which might arise if a proprietary storage format were to be chosen instead. CMS, by contrast, is a popular, well-known, generic and extensible international standard which has been validated in the course of many years in practical use.

Conforming signature generation and verification tools must at least support this format specification. However, any such tool should gracefully handle situations when an unsupported format is encountered, e.g. by simply ignoring these formats. Note that future versions of this specification may define alternative formats for storing PNG digital signatures or message authentication codes.

In any event, the '**external signature**' variant should be chosen in line with section 5.2 of RFC 3852. The encapsulated content type should always be ID data (cf. section 4 of RFC 3852) and the content field value of the encapsulated content information structure must be omitted.

An alternative encapsulated content type could be used in a future update of this specification for indicating a different way of digest calculation, which could then serve to digest a virtual canonical PNG in contrast to digesting the PNG data stream as-is (cf. lenient vs. the current strict approach discussed above). Note that the current specification supports a strict approach only.

An introductory dSIG chunk is used for specifying the digest algorithm(s) used for signature generation or verification of the content in question. A complementary terminating dSIG chunk is used for storing signer information. The embedded signed-data structures must be formatted as follows:

- An introductory dSIG chunk should outline the digest algorithms used for generating or verifying the content in question, whose major part follows the introductory chunk. Certificates and CRLs should be omitted. An empty set of signers should be specified in an introductory dSIG chunk.

- A terminating dSIG chunk should not outline any digest algorithms, the digest algorithms of the corresponding introductory dSIG chunk to be referenced instead. The remaining parts of the CMS structure should be specified as usual.

*Note that a **timestamp signature** can optionally be added (cf. **SigningTime** attribute in section 11.3 of RFC 3852).*

A PNG digital signature provided by this specification proposal on the basis of a CMS embedded in the new **dSIG** extension chunk should be named "**enveloped PNG signature**", in contrast to a possible enveloping PNG signature as described above.

6.2 Processing Notes

An enveloped PNG signature is designed to be generated and verified using streaming capabilities, as is the case for PNG data streams.

A signature is an optional feature of a PNG; by the same token an enveloped PNG signature processor is a supplementary processing device and therefore doesn't check the correctness of the PNG it is processing. As a result, a PNG signature processor acts on top of other PNG processing, thus **delegating** the **checking of** PNG **format correctness** to the underlying PNG processing tools, i.e. viewers and/or editors. An enveloped PNG signature processor only considers correct signature handling, while respecting the entire PNG standard.

Note that this is not a requirement but only a recommendation so as not to duplicate efforts when developing PNG signature processors and not to merge format and signature aspects, which also addresses performance aspects and flexibility for future PNG format extensions. By contrast, a signature processor provider may optionally combine signature generation/verification and PNG format validation.

Hence, there could be PNG data streams with a correct signature but in an incorrect PNG format, e.g. having incorrectly ordered chunks. However, a situation of this type is not a security issue but rather purely a format issue, which could be detected by an ordinary PNG decoder. Generally speaking, one can sign completely meaningless content using a valid digital signature and vice versa.

However, checking the specified order of dSIG chunks is mandatory when verifying digital signatures of a PNG data stream. Signature verification must fail if the specified dSIG chunk order is violated.

Also note that due to the strict approach, signing PNGs should take place after a PNG data stream has been generated. Similarly, signature verification should always be performed of the original PNG data stream before any viewer or editor re-orders chunks due to their own processing strategy.

6.2.1 Signature Generation

An enveloped PNG signature processor should act as a PNG data stream filter in the following manner:

- Read a PNG data stream as the input and calculate the PNG message digest for each algorithm used by the signer(s) over all bytes read in the order they appear:
    - Bypass a leading 8-byte PNG signature of a standalone PNG data stream. Do not take it into account for digest calculation, but write each byte read to the output applying exactly the same order
      Note that "(8-byte) PNG signature" here possesses the semantics specified in the PNG standard, not the semantics of a digital signature.

- For each chunk, read the chunk's type and length bytes and continue reading "blindly" until reaching the end of the chunk (including the checksum).
  Note that chunks are read and processed byte by byte exactly in the order they appear in the input. In addition, length bytes are interpreted as specified in the PNG standard, e.g. network byte order has to be respected. The chunk's checksum does not have to be verified by a signature processor (cf. above-cited note concerning separation of aspects).

  - If the current chunk is not the image trailer, write each byte read to the output applying exactly the same order and use it for digest calculation. Otherwise, buffer the image trailer after using it for digest calculation.

  - If the currently written chunk is the image header generate an introductory **dSIG** chunk as specified above and write it to the output.

- A processor can either retain or discard a previously existing **dSIG** chunk pair, depending on its configuration options. When retained it must be taken into account for digest generation.
  Note that questions on how to deal with previously existing signature information extend beyond the scope of this proposal. A PNG signature processing tool should decide whether to retain or delete previously existing signatures according to its processing policies.

- Calculate the signed-data content type based on the calculated digest(s) and private signing key(s) and other items, e.g. signed attributes. Store the content type (also comprising certificate(s) corresponding to the signing key(s), timestamp signature(s) etc.) to the output, correctly formatted as a terminating **dSIG** chunk.

- Write the buffered image trailer to the output.

Note that when integrating PNG signature generation with PNG editors, the "you should only sign what you see" paradigm should be honored, e.g. by visualizing the PNG to be signed and making the signature generation process transparent to the user.

6.2.2 Signature Verification

An enveloped PNG signature processor should act as a PNG data stream filter in a manner similar to that described in the previous section.

- However, instead of generating **dSIG** chunks, the chunk contents are analyzed and retained.

- When digest calculation is completed after the image trailer has been read, a signature contained in a terminating dSIG chunk has to be verified in the context of the respective newly calculated digest.

- For each attached signature to be verified, do only take the image header, all bytes between the corresponding introductory and terminating dSIG chunks, and the image trailer into account for re-calculating a digest.

- In the case of a standalone PNG, it must be verified that the image header immediately follows a valid 8-byte PNG signature *(cf. above-cited format specification concerning the word "(8-byte) PNG signature" in this context)*.

- The specified order of dSIG chunks must be checked during the process. This also comprises that an introductory dSIG chunk can only be preceded by another introductory dSIG chunk or the image header and that a terminating dSIG chunk can only be followed by another terminating dSIG chunk or the image trailer.

  Note that the previous two rules are crucial for security. They prevent additional information from being introduced into a PNG data stream section, which is different from a dSIG chunk and which is not sealed by a digital signature.

- If verification is successful, additional checks have to be performed according to the available signer information, e.g. PKIX check for certificates (cf. RFC 3280) and timestamp verification.

- If all checks are successful, the overall process is successful with respect to the attached digital signature. Otherwise, it has to fail.

Note that PNG viewers might be made aware of the outcome of an integrated signature verification process so that they could optionally visualize verification results and associated information for the user's convenience.

6.3 Application to Related Formats

This section explains how the dSIG chunk type is applied to other formats of the PNG family.

6.3.1 JNG Data Streams

The JNG application is identical to the PNG application except that the introductory dSIG chunk follows the JHDR chunk instead of IHDR, and an 8-byte JNG signature is present instead of an 8-byte PNG signature.

6.3.2 MNG Data Streams

An entire MNG data stream can be signed by writing an introductory dSIG chunk or chunks immediately following the MHDR chunk and the trailing dSIG chunk or chunks immediately preceding the MEND chunk. Any dSIG chunks found in embedded PNG or JNG data streams are not treated specially, i.e. they are included, when calculating a digest for the entire MNG data stream, and the embedded IEND chunks are not treated specially. An 8-byte MNG signature is present instead of an 8-byte PNG signature and the embedded PNG or JNG data streams do not have 8-byte PNG signatures or 8-byte JNG signatures respectively. The rules concerning the dSIG chunk order have to be applied accordingly as well as the check concerning the 8-byte MNG signature.

7 ADDENDUM

Below, an example of an introductory and a corresponding terminating dSIG chunk is listed using the ASN.1 basic notation (X.680). The actual dSIG content is obtained by applying the ASN.1 encoding rules (X.690).

Syntax productions are referenced by RFC 3280 and RFC 3852. Algorithm identifiers are referenced by RFC 3370.

The annotated example outlines signature information for one signer using sha-1 as the digest algorithm. Some fragments of the terminating dSIG chunk are omitted in order to focus on the essential parts of the structure.

7.1 Introductory dSIG Chunk

```
SEQUENCE {
   OBJECT IDENTIFIER id-signedData (1 2 840 113549 1 7 2)
   [0] {
      SEQUENCE {
         INTEGER 1
         SET { -- the digest algorithms in use
            SEQUENCE {  -- note that parameters are omitted
               OBJECT IDENTIFIER sha-1 (1 3 14 3 2 26)
            }
         }
         SEQUENCE {  -- no encapsulated content due to the external signature option
            OBJECT IDENTIFIER id-data (1 2 840 113549 1 7 1)
         }
         SET { -- signer info is referenced by the terminating dSIG chunk
         }
      }
   }
}
```

7.2 Terminating dSIG Chunk

```
SEQUENCE {
   OBJECT IDENTIFIER id-signedData (1 2 840 113549 1 7 2)
   [0] {
      SEQUENCE {
         INTEGER 1
         SET { -- digest algorithms are referenced by the introductory dSIG chunk
         }
         SEQUENCE {  -- no encapsulated content due to the external signature option
            OBJECT IDENTIFIER id-data (1 2 840 113549 1 7 1)
         }
         [0] {
            -- An IMPLICIT tagged SET comprising:
            --    The signer's end entity certificate referenced by the signer info below
            --    The intermediate certificates for constructing the path to the root:
            --       LuxTrust Normalised CA and LuxTrust root CA
            -- Note that the trusted GTE CyberTrust Global Root would NOT be listed here.
         }
         SET { -- the signer info comprising one signer
            SEQUENCE {
               INTEGER 1   -- the issuer and serial number option is used
               SEQUENCE {
                  SEQUENCE {
                     SET {
                        SEQUENCE {
                           OBJECT IDENTIFIER countryName (2 5 4 6)
                           PrintableString "LU"
                        }
                     }
                     SET {
                        SEQUENCE {
                           OBJECT IDENTIFIER organizationName (2 5 4 10)
                           PrintableString "LuxTrust s.a"
                        }
                     }
                     SET {
                        SEQUENCE {
                           OBJECT IDENTIFIER commonName (2 5 4 3)
                           PrintableString "LuxTrust Normalised CA"
                        }
                     }
                  }
                  INTEGER 1821   -- the serial number of the signer certificate
               }
               SEQUENCE {  -- the digest algorithm used by this signer
                  OBJECT IDENTIFIER sha-1 (1 3 14 3 2 26)
               }
               SEQUENCE {  -- the signature algorithm used by this signer
                  OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
                  NULL
               }
               OCTET STRING   -- the signature value
                  2F 23 82 D2 F3 09 5F B8 0C 58 EB 4E
                  9D BF 89 9A 81 E5 75 C4 91 3D D3 D0
                  D5 7B B6 D5 FE 94 A1 8A AC E3 C4 84
                  F5 CD 60 4E 27 95 F6 CF 00 86 76 75
                  3F 2B F0 E7 D4 02 67 A7 F5 C7 8D 16
                  04 A5 B3 B5 E7 D9 32 F0 24 EF E7 20
                  44 D5 9F 07 C5 53 24 FA CE 01 1D 0F
                  17 13 A7 2A 95 9D 2B E4 03 95 14 0B
                  E9 39 0D BA CE 6E 9C 9E 0C E8 98 E6
                  55 13 D4 68 6F D0 07 D7 A2 B1 62 4C
                  E3 8F AF FD E0 D5 5D C7
            }
         }
      }
   }
}
```