Network Working Group
Request for Comments: 5019
Category: Standards Track

A. Deacon
VeriSign
R. Hurst
Microsoft

September 2007

The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments

Status of This Memo

Abstract

This specification defines a profile of the Online Certificate Status Protocol (OCSP) that addresses the scalability issues inherent when using OCSP in large scale (high volume) Public Key Infrastructure (PKI) environments and/or in PKI environments that require a lightweight solution to minimize communication bandwidth and client-side processing.

Table of Contents

1. Introduction

The Online Certificate Status Protocol [OCSP] specifies a mechanism used to determine the status of digital certificates, in lieu of using Certificate Revocation Lists (CRLs). Since its definition in 1999, it has been deployed in a variety of environments and has proven to be a useful certificate status checking mechanism. (For brevity we refer to OCSP as being used to verify certificate status, but only the revocation status of a certificate is checked via this protocol.)

To date, many OCSP deployments have been used to ensure timely and secure certificate status information for high-value electronic transactions or highly sensitive information, such as in the banking and financial environments. As such, the requirement for an OCSP responder to respond in "real time" (i.e., generating a new OCSP response for each OCSP request) has been important. In addition, these deployments have operated in environments where bandwidth usage is not an issue, and have run on client and server systems where processing power is not constrained.

As the use of PKI continues to grow and move into diverse environments, so does the need for a scalable and cost-effective certificate status mechanism. Although OCSP as currently defined and deployed meets the need of small to medium-sized PKIs that operate on powerful systems on wired networks, there is a limit as to how these OCSP deployments scale from both an efficiency and cost perspective. Mobile environments, where network bandwidth may be at a premium and client-side devices are constrained from a processing point of view, require the careful use of OCSP to minimize bandwidth usage and client-side processing complexity. [OCSPMP]

PKI continues to be deployed into environments where millions if not hundreds of millions of certificates have been issued. In many of these environments, an even larger number of users (also known as relying parties) have the need to ensure that the certificate they are relying upon has not been revoked. As such, it is important that OCSP is used in such a way that ensures the load on OCSP responders and the network infrastructure required to host those responders are kept to a minimum.

This document addresses the scalability issues inherent when using OCSP in PKI environments described above by defining a message profile and clarifying OCSP client and responder behavior that will permit:

1) OCSP response pre-production and distribution.
2) Reduced OCSP message size to lower bandwidth usage.
3) Response message caching both in the network and on the client.

It is intended that the normative requirements defined in this profile will be adopted by OCSP clients and OCSP responders operating in very large-scale (high-volume) PKI environments or PKI environments that require a lightweight solution to minimize bandwidth and client-side processing power (or both), as described above. As OCSP does not have the means to signal responder capabilities within the protocol, clients needing to differentiate between OCSP responses produced by responders conformant with this profile and those that are not need to rely on out-of-band mechanisms to determine when a responder operates according to this profile and, as such, when the requirements of this profile apply. In the case where out-of-band mechanisms may not be available, this profile ensures that interoperability will still occur between a fully conformant OCSP 2560 client and a responder that is operating in a mode as described in this specification.

1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. OCSP Message Profile

This section defines a subset of OCSPRequest and OCSPResponse functionality as defined in [OCSP].

2.1. OCSP Request Profile

2.1.1. OCSPRequest Structure

OCSPRequests conformant to this profile MUST include only one Request in the OCSPRequest.RequestList structure.

Clients MUST use SHA1 as the hashing algorithm for the CertID.issuerNameHash and the CertID.issuerKeyHash values.

Clients MUST NOT include the singleRequestExtensions structure.

Clients SHOULD NOT include the requestExtensions structure. If a requestExtensions structure is included, this profile RECOMMENDS that it contain only the nonce extension (id-pkix-ocsp-nonce). See Section 4 for issues concerning the use of a nonce in high-volume OCSP environments.

2.1.2. Signed OCSPRequests

Clients SHOULD NOT send signed OCSPRequests. Responders MAY ignore the signature on OCSPRequests.

If the OCSPRequest is signed, the client SHALL specify its name in the OCSPRequest.requestorName field; otherwise, clients SHOULD NOT include the requestorName field in the OCSPRequest. OCSP servers MUST be prepared to receive unsigned OCSP requests that contain the requestorName field, but must realize that the provided value is not authenticated.

2.2. OCSP Response Profile

2.2.1. OCSPResponse Structure

Responders MUST generate a BasicOCSPResponse as identified by the id-pkix-ocsp-basic OID. Clients MUST be able to parse and accept a BasicOCSPResponse. OCSPResponses conformant to this profile SHOULD include only one SingleResponse in the ResponseData.responses structure, but MAY include additional SingleResponse elements if necessary to improve response pre-generation performance or cache efficiency.

The responder SHOULD NOT include responseExtensions. As specified in [OCSP], clients MUST ignore unrecognized non-critical responseExtensions in the response.

In the case where a responder does not have the ability to respond to an OCSP request containing a option not supported by the server, it SHOULD return the most complete response it can. For example, in the case where a responder only supports pre-produced responses and does not have the ability to respond to an OCSP request containing a nonce, it SHOULD return a response that does not include a nonce.

Clients SHOULD attempt to process a response even if the response does not include a nonce. See Section 4 for details on validating responses that do not contain a nonce. See also Section 7 for relevant security considerations.

Responders that do not have the ability to respond to OCSP requests that contain an unsupported option such as a nonce MAY forward the request to an OCSP responder capable of doing so.

The responder MAY include the singleResponse.singleResponse extensions structure.

2.2.2. Signed OCSPResponses

Clients MUST validate the signature on the returned OCSPResponse.

If the response is signed by a delegate of the issuing certification authority (CA), a valid responder certificate MUST be referenced in the BasicOCSPResponse.certs structure.

It is RECOMMENDED that the OCSP responder's certificate contain the id-pkix-ocsp-nocheck extension, as defined in [OCSP], to indicate to the client that it need not check the certificate's status. In addition, it is RECOMMENDED that neither an OCSP authorityInfoAccess (AIA) extension nor cRLDistributionPoints (CRLDP) extension be included in the OCSP responder's certificate. Accordingly, the responder's signing certificate SHOULD be relatively short-lived and renewed regularly.

Clients MUST be able to identify OCSP responder certificates using both the byName and byKey ResponseData.ResponderID choices. Responders SHOULD use byKey to further reduce the size of the response in scenarios where reducing bandwidth is an issue.

2.2.3. OCSPResponseStatus Values

As long as the OCSP infrastructure has authoritative records for a particular certificate, an OCSPResponseStatus of "successful" will be returned. When access to authoritative records for a particular certificate is not available, the responder MUST return an OCSPResponseStatus of "unauthorized". As such, this profile extends the RFC 2560 [OCSP] definition of "unauthorized" as follows:

> The response "unauthorized" is returned in cases where the client is not authorized to make this query to this server or the server is not capable of responding authoritatively.

For example, OCSP responders that do not have access to authoritative records for a requested certificate, such as those that generate and distribute OCSP responses in advance and thus do not have the ability to properly respond with a signed "successful" yet "unknown" response, will respond with an OCSPResponseStatus of "unauthorized". Also, in order to ensure the database of revocation information does not grow unbounded over time, the responder MAY remove the status records of expired certificates. Requests from clients for certificates whose record has been removed will result in an OCSPResponseStatus of "unauthorized".

Security considerations regarding the use of unsigned responses are discussed in [OCSP].

2.2.4. thisUpdate, nextUpdate, and producedAt

When pre-producing OCSPResponse messages, the responder MUST set the thisUpdate, nextUpdate, and producedAt times as follows:

thisUpdate	The time at which the status being indicated is known to be correct.
nextUpdate	The time at or before which newer information will be available about the status of the certificate. Responders MUST always include this value to aid in response caching. See Section 6 for additional information on caching.
producedAt	The time at which the OCSP response was signed.

Note: In many cases the value of thisUpdate and producedAt will be the same.

For the purposes of this profile, ASN.1-encoded GeneralizedTime values such as thisUpdate, nextUpdate, and producedAt MUST be expressed Greenwich Mean Time (Zulu) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds.

3. Client Behavior

3.1. OCSP Responder Discovery

Clients MUST support the authorityInfoAccess extension as defined in [PKIX] and MUST recognize the id-ad-ocsp access method. This enables CAs to inform clients how they can contact the OCSP service.

In the case where a client is checking the status of a certificate that contains both an authorityInformationAccess (AIA) extension pointing to an OCSP responder and a cRLDistributionPoints extension pointing to a CRL, the client SHOULD attempt to contact the OCSP responder first. Clients MAY attempt to retrieve the CRL if no OCSPResponse is received from the responder after a locally configured timeout and number of retries.

3.2. Sending an OCSP Request

To avoid needless network traffic, applications MUST verify the signature of signed data before asking an OCSP client to check the status of certificates used to verify the data. If the signature is invalid or the application is not able to verify it, an OCSP check MUST NOT be requested.

Similarly, an application MUST validate the signature on certificates in a chain, before asking an OCSP client to check the status of the certificate. If the certificate signature is invalid or the application is not able to verify it, an OCSP check MUST NOT be requested. Clients SHOULD NOT make a request to check the status of expired certificates.

4. Ensuring an OCSPResponse Is Fresh

In order to ensure that a client does not accept an out-of-date response that indicates a 'good' status when in fact there is a more up-to-date response that specifies the status of 'revoked', a client must ensure the responses they receive are fresh.

In general, two mechanisms are available to clients to ensure a response is fresh. The first uses nonces, and the second is based on time. In order for time-based mechanisms to work, both clients and responders MUST have access to an accurate source of time.

Because this profile specifies that clients SHOULD NOT include a requestExtensions structure in OCSPRequests (see Section 2.1), clients MUST be able to determine OCSPResponse freshness based on an accurate source of time. Clients that opt to include a nonce in the request SHOULD NOT reject a corresponding OCSPResponse solely on the basis of the nonexistent expected nonce, but MUST fall back to validating the OCSPResponse based on time.

Clients that do not include a nonce in the request MUST ignore any nonce that may be present in the response.

Clients MUST check for the existence of the nextUpdate field and MUST ensure the current time, expressed in GMT time as described in Section 2.2.4, falls between the thisUpdate and nextUpdate times. If the nextUpdate field is absent, the client MUST reject the response.

If the nextUpdate field is present, the client MUST ensure that it is not earlier than the current time. If the current time on the client is later than the time specified in the nextUpdate field, the client MUST reject the response as stale. Clients MAY allow configuration of a small tolerance period for acceptance of responses after nextUpdate to handle minor clock differences relative to responders and caches. This tolerance period should be chosen based on the accuracy and precision of time synchronization technology available to the calling application environment. For example, Internet peers with low latency connections typically expect NTP time synchronization to keep them accurate within parts of a second; higher latency environments or where an NTP analogue is not available may have to be more liberal in their tolerance.

See the security considerations in Section 7 for additional details on replay and man-in-the-middle attacks.

5. Transport Profile

The OCSP responder MUST support requests and responses over HTTP. When sending requests that are less than or equal to 255 bytes in total (after encoding) including the scheme and delimiters (http://), server name and base64-encoded OCSPRequest structure, clients MUST use the GET method (to enable OCSP response caching). OCSP requests larger than 255 bytes SHOULD be submitted using the POST method. In all cases, clients MUST follow the descriptions in A.1.1 of [OCSP] when constructing these messages.

When constructing a GET message, OCSP clients MUST base64 encode the OCSPRequest structure and append it to the URI specified in the AIA extension [PKIX]. Clients MUST NOT include CR or LF characters in the base64-encoded string. Clients MUST properly URL-encode the base64 encoded OCSPRequest. For example:

```
http://ocsp.example.com/MEowSDBGMEQwQjAKBggqhkiG9w0CBQQQ7sp6GTKpL
2dAdeGaW267owQQqInESWQD0mGeBArSgv%2FBWQIQLJx%2Fg9xF8oySYzol80Mbpg
%3D%3D
```

In response to properly formatted OCSPRequests that are cachable (i.e., responses that contain a nextUpdate value), the responder will include the binary value of the DER encoding of the OCSPResponse preceded by the following HTTP [HTTP] headers.

```
content-type: application/ocsp-response
content-length: <OCSP response length>
last-modified: <producedAt [HTTP] date>
ETag: "<strong validator>"
expires: <nextUpdate [HTTP] date>
cache-control: max-age=<n>, public, no-transform, must-revalidate
date: <current [HTTP] date>
```

See Section 6.2 for details on the use of these headers.

6. Caching Recommendations

The ability to cache OCSP responses throughout the network is an important factor in high volume OCSP deployments. This section discusses the recommended caching behavior of OCSP clients and HTTP proxies and the steps that should be taken to minimize the number of times that OCSP clients "hit the wire". In addition, the concept of including OCSP responses in protocol exchanges (aka stapling or piggybacking), such as has been defined in TLS, is also discussed.

6.1. Caching at the Client

To minimize bandwidth usage, clients MUST locally cache authoritative OCSP responses (i.e., a response with a signature that has been successfully validated and that indicate an OCSPResponseStatus of 'successful').

Most OCSP clients will send OCSPRequests at or near the nextUpdate time (when a cached response expires). To avoid large spikes in responder load that might occur when many clients refresh cached responses for a popular certificate, responders MAY indicate when the client should fetch an updated OCSP response by using the cache-control:max-age directive. Clients SHOULD fetch the updated OCSP Response on or after the max-age time. To ensure that clients receive an updated OCSP response, OCSP responders MUST refresh the OCSP response before the max-age time.

6.2. HTTP Proxies

The responder SHOULD set the HTTP headers of the OCSP response in such a way as to allow for the intelligent use of intermediate HTTP proxy servers. See [HTTP] for the full definition of these headers and the proper format of any date and time values.

HTTP Header	Description
date	The date and time at which the OCSP server generated the HTTP response.
last-modified	This value specifies the date and time at which the OCSP responder last modified the response. This date and time will be the same as the thisUpdate timestamp in the request itself.
expires	Specifies how long the response is considered fresh. This date and time will be the same as the nextUpdate timestamp in the OCSP response itself.
ETag	A string that identifies a particular version of the associated data. This profile RECOMMENDS that the ETag value be the ASCII HEX representation of the SHA1 hash of the OCSPResponse structure.
cache-control	Contains a number of caching directives.
	* max-age=<n> -where n is a time value later than thisUpdate but earlier than nextUpdate.

* public -makes normally uncachable response cachable by both shared and nonshared caches.

* no-transform -specifies that a proxy cache cannot change the type, length, or encoding of the object content.

* must-revalidate -prevents caches from intentionally returning stale responses.

OCSP responders MUST NOT include a "Pragma: no-cache", "Cache-Control: no-cache", or "Cache-Control: no-store" header in authoritative OCSP responses.

OCSP responders SHOULD include one or more of these headers in non-authoritative OCSP responses.

For example, assume that an OCSP response has the following timestamp values:

```
thisUpdate = May 1, 2005  01:00:00 GMT
nextUpdate = May 3, 2005 01:00:00 GMT
productedAt = May 1, 2005 01:00:00 GMT
```

and that an OCSP client requests the response on May 2, 2005 01:00:00 GMT. In this scenario, the HTTP response may look like this:

```
content-type: application/ocsp-response
content-length: 1000
date: Fri, 02 May 2005 01:00:00 GMT
last-modified: Thu, 01 May 2005 01:00:00 GMT
ETag: "c66c0341abd7b9346321d5470fd0ec7cc4dae713"
expires: Sat, 03 May 2005 01:00:00 GMT
cache-control: max-age=86000,public,no-transform,must-revalidate
<...>
```

OCSP clients MUST NOT include a no-cache header in OCSP request messages, unless the client encounters an expired response which may be a result of an intermediate proxy caching stale data. In this situation, clients SHOULD resend the request specifying that proxies should be bypassed by including an appropriate HTTP header in the request (i.e., Pragma: no-cache or Cache-Control: no-cache).

6.3. Caching at Servers

In some scenarios, it is advantageous to include OCSP response information within the protocol being utilized between the client and server. Including OCSP responses in this manner has a few attractive effects.

First, it allows for the caching of OCSP responses on the server, thus lowering the number of hits to the OCSP responder.

Second, it enables certificate validation in the event the client is not connected to a network and thus eliminates the need for clients to establish a new HTTP session with the responder.

Third, it reduces the number of round trips the client needs to make in order to complete a handshake.

Fourth, it simplifies the client-side OCSP implementation by enabling a situation where the client need only the ability to parse and recognize OCSP responses.

This functionality has been specified as an extension to the TLS [TLS] protocol in Section 3.6 [TLSEXT], but can be applied to any client-server protocol.

This profile RECOMMENDS that both TLS clients and servers implement the certificate status request extension mechanism for TLS.

Further information regarding caching issues can be obtained from RFC 3143 [RFC3143].

7. Security Considerations

The following considerations apply in addition to the security considerations addressed in Section 5 of [OCSP].

7.1. Replay Attacks

Because the use of nonces in this profile is optional, there is a possibility that an out of date OCSP response could be replayed, thus causing a client to accept a good response when in fact there is a more up-to-date response that specifies the status of revoked. In order to mitigate this attack, clients MUST have access to an accurate source of time and ensure that the OCSP responses they receive are sufficiently fresh.

Clients that do not have an accurate source of date and time are vulnerable to service disruption. For example, a client with a sufficiently fast clock may reject a fresh OCSP response. Similarly a client with a sufficiently slow clock may incorrectly accept expired valid responses for certificates that may in fact be revoked.

Future versions of the OCSP protocol may provide a way for the client to know whether the server supports nonces or does not support nonces. If a client can determine that the server supports nonces, it MUST reject a reply that does not contain an expected nonce. Otherwise, clients that opt to include a nonce in the request SHOULD NOT reject a corresponding OCSPResponse solely on the basis of the nonexistent expected nonce, but MUST fall back to validating the OCSPResponse based on time.

7.2. Man-in-the-Middle Attacks

To mitigate risk associated with this class of attack, the client must properly validate the signature on the response.

The use of signed responses in OCSP serves to authenticate the identity of the OCSP responder and to verify that it is authorized to sign responses on the CA's behalf.

Clients MUST ensure that they are communicating with an authorized responder by the rules described in [OCSP], Section 4.2.2.2.

7.3. Impersonation Attacks

The use of signed responses in OCSP serves to authenticate the identity of OCSP responder.

As detailed in [OCSP], clients must properly validate the signature of the OCSP response and the signature on the OCSP response signer certificate to ensure an authorized responder created it.

7.4. Denial-of-Service Attacks

OCSP responders should take measures to prevent or mitigate denial-of-service attacks. As this profile specifies the use of unsigned OCSPRequests, access to the responder may be implicitly given to everyone who can send a request to a responder, and thus the ability to mount a denial-of-service attack via a flood of requests may be greater. For example, a responder could limit the rate of incoming requests from a particular IP address if questionable behavior is detected.

7.5. Modification of HTTP Headers

Values included in HTTP headers, as described in Sections 5 and 6, are not cryptographically protected; they may be manipulated by an attacker. Clients SHOULD use these values for caching guidance only and ultimately SHOULD rely only on the values present in the signed OCSPResponse. Clients SHOULD NOT rely on cached responses beyond the nextUpdate time.

7.6. Request Authentication and Authorization

The suggested use of unsigned requests in this environment removes an option that allows the responder to determine the authenticity of incoming request. Thus, access to the responder may be implicitly given to everyone who can send a request to a responder. Environments where explicit authorization to access the OCSP responder is necessary can utilize other mechanisms to authenticate requestors or restrict or meter service.

8. Acknowledgements

The authors wish to thank Magnus Nystrom of RSA Security, Inc., Jagjeet Sondh of Vodafone Group R&D, and David Engberg of CoreStreet, Ltd. for their contributions to this specification.

9. References

9.1. Normative References

[HTTP] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[OCSP] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[PKIX] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security Protocol Version 1.1", RFC 4346, April 2006.

[TLSEXT] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

9.2. Informative References

[OCSPMP] "OCSP Mobile Profile V1.0", Open Mobile Alliance, www.openmobilealliance.org.

[RFC3143] Cooper, I. and J. Dilley, "Known HTTP Proxy/Caching Problems", RFC 3143, June 2001.

Appendix A. Example OCSP Messages

A.1. OCSP Request

```
   SEQUENCE {
      SEQUENCE {
        SEQUENCE {
          SEQUENCE {
            SEQUENCE {
              SEQUENCE {
                SEQUENCE {
                  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
                  NULL
                  }
                OCTET STRING
                  C0 FE 02 78 FC 99 18 88 91 B3 F2 12 E9 C7 E1 B2
                  1A B7 BF C0
                OCTET STRING
                  0D FC 1D F0 A9 E0 F0 1C E7 F2 B2 13 17 7E 6F 8D
                  15 7C D4 F6
                INTEGER
                  09 34 23 72 E2 3A EF 46 7C 83 2D 07 F8 DC 22 BA
                }
              }
            }
          }
        }
      }
```

A.2. OCSP Response

```
   SEQUENCE {
      ENUMERATED 0
      [0] {
        SEQUENCE {
          OBJECT IDENTIFIER ocspBasic (1 3 6 1 5 5 7 48 1 1)
          OCTET STRING, encapsulates {
            SEQUENCE {
              SEQUENCE {
                [0] {
                  INTEGER 0
                  }
                [1] {
                  SEQUENCE {
                    SET {
                      SEQUENCE {
                        OBJECT IDENTIFIER organizationName (2 5 4 10)
                        PrintableString 'Example Trust Network'
                        }
                      }
```

```
                SET {
                  SEQUENCE {
                    OBJECT IDENTIFIER
                      organizationalUnitName (2 5 4 11)
                    PrintableString 'Example, Inc.'
                    }
                  }
                SET {
                  SEQUENCE {
                    OBJECT IDENTIFIER
                      organizationalUnitName (2 5 4 11)
                    PrintableString
                  'Example OCSP Responder'
                    }
                  }
                }
              }
            GeneralizedTime 07/11/2005 23:52:44 GMT
            SEQUENCE {
              SEQUENCE {
                SEQUENCE {
                  SEQUENCE {
                    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
                    NULL
                    }
                  OCTET STRING
                  C0 FE 02 78 FC 99 18 88 91 B3 F2 12 E9 C7 E1 B2
                  1A B7 BF C0
                  OCTET STRING
                  0D FC 1D F0 A9 E0 F0 1C E7 F2 B2 13 17 7E 6F 8D
                  15 7C D4 F6
                  INTEGER
                  09 34 23 72 E2 3A EF 46 7C 83 2D 07 F8 DC 22 BA
                  }
                [0]
                  Error: Object has zero length.
                  GeneralizedTime 07/11/2005 23:52:44 GMT
                [0] {
                  GeneralizedTime 14/11/2005 23:52:44 GMT
                  }
                }
              }
            }
          SEQUENCE {
            OBJECT IDENTIFIER
              sha1withRSAEncryption (1 2 840 113549 1 1 5)
            NULL
            }
```

```
            BIT STRING
              0E 9F F0 52 B1 A7 42 B8 6E C1 35 E1 0E D5 A9 E2
              F5 C5 3C 16 B1 A3 A7 A2 03 8A 2B 4D 2C F1 B4 98
              8E 19 DB BA 1E 1E 72 FF 32 F4 44 E0 B2 77 1C D7
              3C 9E 78 F3 D1 82 68 86 63 12 7F A4 6F F0 4D 84
              EA F8 E2 F7 5D E3 48 44 57 28 80 C7 57 3C FE E1
              42 0E 5E 17 FC 60 D8 05 D9 EF E2 53 E7 AB 7F 3A
              A8 84 AA 5E 46 5B E7 B8 1F C6 B1 35 AD FF D1 CC
              BA 58 7D E8 29 60 79 F7 41 02 EA E0 82 0E A6 30
            [0] {
              SEQUENCE {
                SEQUENCE {
                  SEQUENCE {
                    [0] {
                      INTEGER 2
                      }
                    INTEGER
                    49 4A 02 37 1B 1E 70 67 41 6C 9F 06 2F D8 FE DA
                    SEQUENCE {
                      OBJECT IDENTIFIER
                        sha1withRSAEncryption (1 2 840 113549 1 1 5)
                      NULL
                      }
                    SEQUENCE {
                      SET {
                        SEQUENCE {
                          OBJECT IDENTIFIER
                            organizationName (2 5 4 10)
                          PrintableString 'Example Trust Network'
                          }
                        }
                      SET {
                        SEQUENCE {
                          OBJECT IDENTIFIER
                            organizationalUnitName (2 5 4 11)
                          PrintableString 'Example, Inc.'
                          }
                        }
                      SET {
                        SEQUENCE {
                          OBJECT IDENTIFIER
                            organizationalUnitName (2 5 4 11)
                          PrintableString
                             'Example CA'
                          }
                        }
                      }
                    SEQUENCE {
```

```
      UTCTime 08/10/2005 00:00:00 GMT
      UTCTime 06/01/2006 23:59:59 GMT
      }
    SEQUENCE {
      SET {
        SEQUENCE {
          OBJECT IDENTIFIER
            organizationName (2 5 4 10)
          PrintableString 'Example Trust Network'
          }
        }
      SET {
        SEQUENCE {
          OBJECT IDENTIFIER
            organizationalUnitName (2 5 4 11)
          PrintableString 'Example, Inc.'
          }
        }
      SET {
        SEQUENCE {
          OBJECT IDENTIFIER
            organizationalUnitName (2 5 4 11)
          PrintableString
            'Example OCSP Responder'
          }
        }
      }
    SEQUENCE {
      SEQUENCE {
        OBJECT IDENTIFIER
          rsaEncryption (1 2 840 113549 1 1 1)
        NULL
        }
      BIT STRING, encapsulates {
        SEQUENCE {
          INTEGER
    00 AF C9 7A F5 09 CA D1 08 8C 82 6D AC D9 63 4D
    D2 64 17 79 CB 1E 1C 1C 0C 6E 28 56 B5 16 4A 4A
    00 1A C1 B0 74 D7 B4 55 9D 2A 99 1F 0E 4A E3 5F
    81 AF 8D 07 23 C3 30 28 61 3F B0 C8 1D 4E A8 9C
    A6 32 B4 D2 63 EC F7 C1 55 7A 73 2A 51 99 00 D5
    0F B2 4E 11 5B 83 55 83 4C 0E DD 12 0C BD 7E 41
    04 3F 5F D9 2A 65 88 3C 2A BA 20 76 1D 1F 59 3E
    D1 85 F7 4B E2 81 50 9C 78 96 1B 37 73 12 1A D2
            [ Another 1 bytes skipped ]
          INTEGER 65537
          }
        }
```

```
      }
    [3] {
      SEQUENCE {
        SEQUENCE {
          OBJECT IDENTIFIER
            basicConstraints (2 5 29 19)
          OCTET STRING, encapsulates {
            SEQUENCE {}
            }
          }
        SEQUENCE {
          OBJECT IDENTIFIER extKeyUsage (2 5 29 37)
          OCTET STRING, encapsulates {
            SEQUENCE {
              OBJECT IDENTIFIER
                ocspSigning (1 3 6 1 5 5 7 3 9)
              }
            }
          }
        SEQUENCE {
          OBJECT IDENTIFIER keyUsage (2 5 29 15)
          OCTET STRING, encapsulates {
            BIT STRING 7 unused bits
              '1'B (bit 0)
            }
          }
        SEQUENCE {
          OBJECT IDENTIFIER
            ocspNoCheck (1 3 6 1 5 5 7 48 1 5)
          OCTET STRING, encapsulates {
            NULL
            }
          }
        }
      }
  SEQUENCE {
    OBJECT IDENTIFIER
      sha1withRSAEncryption (1 2 840 113549 1 1 5)
    NULL
    }
  BIT STRING
    3A 68 5F 6A F8 87 36 4A E2 22 46 5C C8 F5 0E CE
    1A FA F2 25 E1 51 AB 37 BE D4 10 C8 15 93 39 73
    C8 59 0F F0 39 67 29 C2 60 20 F7 3F FE A0 37 AB
    80 0B F9 3D 38 D4 48 67 E4 FA FD 4E 12 BF 55 29
    14 E9 CC CB DD 13 82 E9 C4 4D D3 85 33 C1 35 E5
    8F 38 01 A7 F7 FD EB CD DE F2 F7 85 86 AE E3 1B
```

```
                    9C FD 1D 07 E5 28 F2 A0 5E AC BF 9E 0B 34 A1 B4
                    3A A9 0E C5 8A 34 3F 65 D3 10 63 A4 5E 21 71 5A
                  }
                }
              }
            }
          }
        }
      }
    }
```

Authors' Addresses

Alex Deacon
VeriSign, Inc.
487 E. Middlefield Road
Mountain View, CA 94043
USA

Phone: 1-650-426-3478
EMail: alex@verisign.com

Ryan Hurst
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

Phone: 1-425-707-8979
EMail: rmh@microsoft.com

Full Copyright Statement

Intellectual Property