Network Working Group
draft-gutmann-cms-rtcs-01.txt
Expires September 2004

P.Gutmann
University of Auckland

March 2004

Real-time Certificate Status Facility for CMS - (RTCS)

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

1. Abstract

This document describes how the Cryptographic Message Syntax may be used for communicating certificate status information in a manner suitable for use with CMS data and CMS-related messaging mechanisms such as S/MIME.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in

uppercase, as shown) are to be interpreted as described in [RFC2119], except when they appear in ASN.1 constructs, in which case they follow [X.680]

2. Problem analysis

This section examines the problems that need to be solved by the protocol, and provides a rationale for design decisions. Section 3 defines the protocol based on the design decisions.

2.1 Overview

When the OCSP protocol was defined, the design was based on full compatibility with CRL-based mechanisms and the use of a nonstandard message format incompatible with the Cryptographic Message Syntax. This requires the use of a complex means of certificate identification that has resulted in interoperability problems among implementations, a design unsuited for high-throughput, real-time operation, the inability to provide an unambiguous certificate status response (the only thing that a CRL can say with certainty is "revoked"), and an online responder tied to an offline mechanism (some CAs issue CRLs only once or twice a day, even though they have an online, real-time certificate store available). A more practical problem is that it makes it impossible to implement an OCSP responder using a standard CMS toolkit or implementation, or not based on CRLs, for example one that consults a certificate database or in-memory hash table to determine the presence or absence of a valid certificate.

Just as the original OCSP responses were designed for completely CRL-compatible operation, this document specifies a response type that is designed for real-time status operation, providing a response not from a stored CRL using CRL-only mechanisms but directly from a live certificate store or in-memory hash table. This allows the responder to provide extended information not possible with CRLs, combined a high level of performance and CMS-compatibility not possible with the original OCSP design.

In abstract terms, the responder is providing an implementation of an authenticated dictionary D that responds to membership queries from relying parties. An OCSP responder answers the question "Is x excluded from D?", while an RTCS responder answers the question "Is x present in D?".

When returning a response, the responder is merely indicating that the queried certificate is currently present in its set of valid certificates in a standard CMS-compatible manner. It is purely an authenticated dictionary service and does not verify the certificate in any way. Relying parties requiring external verification services should use the PKIX standard

mechanisms for this [RFC3379] and not RTCS. Specifically, RTCS does not provide, and should not be assumed to provide, any of the functionality of DPD/DPV. It is purely a mechanism for running a high-performance CMS-compatible certificate status responder directly from a CA certificate store/in-memory table.

Some of the issues that need to be addressed in order to perform this task are covered in the following subsections.

2.2 Use of standard/flexible data formats

The format used for OCSP responses is an incomplete reinvention of the standard CMS format that lacks a number of CMS features, leading to various implementation/deployment difficulties. For example, some responders need to provide confidentiality protection for their responses, since returning an indication that a certificate has been revoked may be interpreted as a statement about the veracity of the organisation that owns the certificate, leading to potential liability concerns (the same problem is faced by some CAs, who have to password-protect or encrypt their CRLs). Similarly, some users require confidentiality protection on requests in order to prevent traffic analysis by outsiders, for the same reasons that protection of responses is required. These operations are trivial to implement using the standard CMS format, but impossible with the OCSP reinvention of CMS, requiring the use of ad-hoc/proprietary extensions.

Responders that operate in resource-constrained environments (see section 2.6) or that require high-throughput operation (see section 2.5) may choose to authenticate their responses with a low-overhead MAC rather than a high-overhead signature. Again, this is impossible with the OCSP format but trivial with CMS.

Finally, CMS is the standard format for signed/encrypted/MAC'ed data. Using this format rather than an incompatible reinvention of the format allows for simple implementations based on existing code. In the case of the OCSP specification, more than half the ASN.1 is dedicated to reinventing the CMS message format; omitting this unnecessary step considerably simplifies the specification and the task of implementation.

2.3 Certificate identification

OCSP defines a complex certificate identifier that takes portions of the certificate, hashes some (making reference to the original value impossible), doesn't hash others, and even requires a hash of data from other certificates to be included as part of the identifier, making it impossible to query the

status of a single, standalone certificate. The OCSP identifier is also incompatible with all existing identifiers, including the one traditionally used by CMS. Real-world experience has shown that implementors have considerable difficulty with this identifier, leading to interoperability problems among implementations.

RTCS should therefore provide a simple, widely-accepted, universally-applicable identifier for all certificates, regardless of their schema or encoding. For compatibility with legacy implementations, it also provides a CRL-compatible identifier, although there are some caveats attached to its use (see section 3.1).

2.4 Returned status value

Because of its CRL-based origins, OCSP can only return a negative response. For example, when fed a freshly-issued certificate and asked "Is this a valid certificate", it can't say "Yes" (a CRL can only answer "revoked"), and when fed an Excel spreadsheet it can't say "No" (the spreadsheet won't be present in any CRL). More seriously, CRLs and OCSP are incapable of dealing with a manufactured-certificate attack in which an attacker issues a certificate claiming to be from a legitimate CA (since the legitimate CA never issued it, it won't be in its CRL, therefore a blacklist-based system can't report the certificate as invalid). This attack is made significantly easier by the implicit universal cross-certification present in many web browsers, where any CA can usurp any other CA's certificates. Even without this universal cross-certification mechanism, standard practice for browsers when encountering an unknown certificate is to enquire of the user "... do you want to trust <company_name>?", where company_name is the company running the site the user is connecting to. Since the certificate is a manufactured certificate being used in a MITM attack, it won't be present on the CRL of the real CA, and since it corresponds to the site that the user is connecting to, they are unlikely to reject it.

The unclear-status problem interacts badly with the one in section 2.3 in that an unknown response could mean anything from "I couldn't find a CRL for this certificate" to "I don't know the status of this certificate" to "This may well be a non-revoked certificate but your software and mine disagree over how to generate the identifier", and there is no way to determine what the actual problem is.

To resolve this issue, RTCS should provide a clear, unambiguous response to any query, either "This certificate is definitely valid right now", "This certificate is definitely not valid right now", or "The object you have queried doesn't exist" (standard OCSP can't do any of these).

2.5 Lightweight/realtime operation

OCSP requires that every response from a responder be authenticated with a signature, whether this is appropriate or not. In cases where high transaction volumes need to be handled, the overhead of having to sign each transaction can be prohibitive (this is one of the few areas in which offline CRLs actually have an advantage over online queries), resulting in scalability and deployment problems. This lack of scalability is sever enough that several vendors have resorted to removing replay protection from the protocol (making it possible for an attacker to undetectably replay old responses) because this is the only way to get OCSP to scale.

In many cases, a lightweight MAC (in other words CMS AuthenticatedData) is all that's required to authenticate a response, and where alternative security measures are used (for example IPsec or the use of a physically secure network), no explicit authentication (in other words CMS Data) may be necessary, allowing RTCS queries to proceed at network link/server turnaround speed. When parties have a long-term relationship (examples being OCSP access concentrators or in Identrus terminology transaction coordinators) and perform large numbers of transactions, authenticating the transactions via MACs makes more sense than signing each one. Similarly, when the producer and consumer of the information are on opposite sides of the same server room, there is little need for high-overhead signatures on each message.

A design goal of RTCS then is that, living up to its name, it must be able to provide high-throughput, low-overhead realtime service to relying parties, via the flexible selection of data formats provided by CMS.

2.6 Use in constrained environments

As an extension of the previous requirement, the protocol should be capable of running in resource- or bandwidth-constrained environments. In its most minimal implementation, RTCS has a small number of fixed-length fields, allowing it to be used by dropping data into pre-generated CMS PDUs. The very small message size and minimal processing requirements make it ideal for use with mobile and remote devices, high-volume transaction systems, and in other constrained environments.

2.7 Reliance on synchronised clocks

OCSP uses timestamps for all responses, assuming that the relying party and responder somehow have perfectly synchronised clocks. This is rarely the case, with systems having been encounted with clocks that are as much as decades out of sync [Gutmann]. RTCS, almost by definition, does not rely on

synchronised clocks for its operation, although it can make use of them when they are available.

3. RTCS

RTCS is designed to provide online, real-time certificate status information using the CMS message format, in a manner that meets all of the design goals given in section 2.

3.1 RTCS requests

An RTCS request consists of an indication of the type of reply required from the responder and a list of certificates for which information is required:

```
RtcsRequest ::= CMS { RtcsRequests IDENTIFIED BY rtcsRequest }

rtcsRequest OBJECT IDENTIFIER { 1 3 6 1 4 1 3029 4 1 4 }

RtcsResponseType OBJECT IDENTIFIER ::= {
  rtcsBasic { 1 3 6 1 4 1 3029 4 1 5 },
  rtcsExtended { 1 3 6 1 4 1 3029 4 1 6 },
  ...
  }

RtcsRequests ::= SEQUENCE {
  responseType     RtcsResponseType DEFAULT rtcsBasic,
  requests         SEQUENCE OF RtcsRequestInfo,
  attributes       Attributes OPTIONAL
  }
```

responseType is the type of response requested from the responder. If the responder cannot provide the requested response type, if MUST return an rtcsBasic response instead.

requests is the sequence of identifiers for the certificates being queried.

attributes is normally unnecessary, but is provided for use when the encapsulating CMS type doesn't provide for the conveyance of attributes. If the encapsulating CMS type supports the conveyance of attributes, they MUST be included in the CMS encapsulation rather than in the RTCS request

attributes field.

As the ASN.1 above indicates, any of the standard CMS encapsulation types may be used to contain the RTCS request, providing authentication and/or confidentiality as required.

```
RtcsRequestInfo ::= SEQUENCE {
  certHash         OtherHash,
  legacyID         IssuerAndSerialNumber OPTIONAL
  }
```

certHash is an SHA-1 hash of the certificate. Almost everything implements this (variously as "fingerprint" or "thumbprint" or under some similar name), the ID type is widely recognised, and interoperability/correctness checking is trivial to achieve. The full definition of OtherHash is given in [RFC3126], however as used here it SHOULD be regarded as a pure sha1Hash:

```
sha1Hash ::= OCTET STRING SIZE(20)
```

legacyID is provided when backwards-compatibility with CRL-based legacy implementations or implementations that only support the traditional CMS certificate identifier are required. The full definition is given in [RFC3369]. This identifier is the standard certificate identifier for CMS and S/MIME, and may be trivially generated from any X.509 certificate. This identifier MUST be included when it is known that the responder is a legacy implementation, and SHOULD be used when the client is unclear as to the status of the responder. It MAY be omitted in resource-constrained environments, or when the client knows that the responder is capable of handling the certHash. See the security considerations for a note on this identifier type.

3.1.1 Additional requirements

Since RTCS doesn't depend on synchronised clocks, implementations operating in environments where replay attacks are a concern MUST use the randomNonce extension [RFC2985] to ensure freshness of replies. For the avoidance of any doubt, when no replay protection is required (for example when other security measures such as link encryption/authentication or a physically secure link are in place), no nonce is required. When replay protection is required and the request or response is communicated using a CMS data type with no provision for communicating attributes (for example CMS Data or CMS EnvelopedData), the nonce MUST be communicated in the rtcsRequest attributes field. If the request or response is communicated using a CMS data type that supports the communication of attributes (for example CMS SignedData or CMS

AuthenticatedData), the nonce MUST be communicated as a CMS authenticated attribute. RTCS implementations MUST support Data requests and SignedData responses, SHOULD support SignedData requests, and MAY support other standard CMS message types and combinations such as Data requests and responses or EncrypteData requests and responses.

3.1.2 Zone transfers

Sometimes it may be desirable for a client to obtain all of the information held by an RTCS responder, for example for mirroring/replication purposes. To provide for RTCS zone transfers, a certHash of all zero bits is used to indicate that it the responder should send information on all certificates that it is authoritative for, and a certHash of all one bits is used to indicate that the responder should send information on all certificates. Responders MAY implemenent this facility by checking for these special certHash values and responding appropriately.

Since zone transfers can consume significant resources, responders SHOULD enforce some form of security controls on these requests, for example by requiring them to be authenticated via CMS SignedData or AuthenticatedData, or by only allowing them when the request is conveyed via a trusted/secure link.

3.1.3 Implementation notes and rationale

The certHash identifier meets the requirements in section 2.3 (use of a widely-accepted, simple, universal identifier for certificates) and section 2 (ability to be used in a constrained environment).

The certificate hash is a universal identifier in that it doesn't care what type or version of certificate is used, whether it's encoded in DER or BER or XER, or whether the certificate even has a DN. It works with X.509 certificates (v1, v2, or v3) with or without extensions, X.509 attribute certificates (v1 or v2), special-case certificates such as X9.68 domain certificates, and any other certificate or certificate-like object that may appear in the future. The hash does not require writing, testing, documenting and maintaining the programming logic needed to handle DN complexity, and is immune to the DN-based problems that affect OCSP.

The backup legacyID may be used with CRL-based legacy implementations, or in situations where the certificate store is implemented as an LDAP directory that identifies certificates by DN. This ensures full backwards compatibility with CRL-based implementations, and an ability to function with LDAP directories that isn't possible with OCSP since it destroys the DN by hashing it.

Implementations are required to support at least the rtcsBasic response type, falling back to this type if the requested type can't be provided. This ensures that at least some form of response is always provided, even if it consists only of an indication that no definitive status is available.

A resource- or bandwidth-constrained environment may use a pre-generated RTCS query and copy the certHash directly into a fixed location in the query. This makes RTCS amenable for use in crypto tokens or mobile devices or high-volume transaction systems that don't have the resources to handle a full implementation and that merely populate a pre-generated query with a fresh nonce and 20-byte certHash.

The full definition of OtherHash, from [RFC3126], is:

```
  OtherHash ::= CHOICE {
    sha1Hash          OCTET STRING SIZE(20),
    otherHash         OtherHashAlgAndValue
    }

  OtherHashAlgAndValue ::= SEQUENCE {
    hashAlgorithm     AlgorithmIdentifier,
    hashValue         OCTET STRING
    }
```

The intent here is that if a weakness is found in SHA-1, an alternative hash algorithm may be substituted in its place. Since every Internet security protocol ever created would require replacing if SHA-1 was broken this is probably a lesser concern, but an alternative is provided here anyway. In standard usage the above simplies to a straight SHA-1 hash.

A pure boolean response (corresponding to a present/absent check in the authenticated dictionary) provides for considerable efficiency improvments on the server, since such a check can be implemented using a mechanism such as a (suitably tuned) Bloom filter that takes advantage of the fact that the query material is already pre-hashed by the client. Clients should submit basic queries (which allow for a simple boolean response) if possible, rather than asking for a full response every time simply because it's available. Conversely, servers may perform a simple boolean lookup initially on the assumption that the majority of certificates being queried will be valid, and only fall back to a more time-consuming full lookup if the initial boolean lookup returns a response of 'false'.

RTCS zone transfers work in the same way as, and have the same implications as, DNS zone transfers. Any standard reference on DNS operations or DNS security will contain further details on this issue. A typical configuration would contain an RTCS primary and secondary responder, just as with DNS servers, with synchronisation being performed via RTCS zone transfers. An alternative strategy, used by some DNS servers, is one in which one server is used for updates and one or more further servers, updated via zone transfers, are used to respond to queries. Again, standard DNS practice provides guidance on building a high-availability, fault-tolerant system on this basis. In fact a scheme similar to RTCS that uses DNS instead of its own specific protocol has already been in general use in Europe as part of the X-Road project.

The means of requesting an RTCS zone transfer has been chosen so that a client or server implemenetation can choose not to provide for zone transfers without any special-case handling for requests or responses. A client requesting a zone transfer from a responder that doesn't support them will receive a no-information response as it would when querying a nonexistant certificate.

3.2 RTCS response

RTCS provides for two response types, a basic response when only a simple yes/no status is required, and a full response when extended information is required.

```
  RTCSRESPONSE ::= TYPE-IDENTIFIER


  RtcsResponse ::= CMS { RTCSRESPONSE.&Type({ RtcsResponseTypes }{ @.type-id })


  RtcsResponseTypes RTCSRESPONSE ::= {
    rtcsResponseBasic | rtcsResponseExtended,
    ...
    }


  rtcsResponseBasic RTCSRESPONSE ::= {
    SYNTAX RtcsResponsesBasic ID { rtcsBasic }
    }


  rtcsResponseExtended RTCSRESPONSE ::= {
    SYNTAX RtcsResponsesExtended ID { rtcsExtended }
    }
```

As the ASN.1 above indicates, any of the standard CMS encapsulation types may be used to contain the RTCS response, providing authentication and/or confidentiality as required.

3.2.1 RTCS basic response

This is a straightforward yes/no response type:

```
  RtcsResponsesBasic ::= SEQUENCE OF RtcsResponseBasic


  RtcsResponseBasic ::= SEQUENCE {
    certHash          OtherHash,
    status            BOOLEAN
    }
```

A returned value 'true' indicates that the certificate is valid right now. This is a clear, unambiguous response that is useful for relying parties who, having a certificate at hand, simply want to know whether they can safely use it or not, and no more. A returned value 'false' indicates that the certificate is not valid right now, either because it has been explicitly rendered invalid in some manner (for example by being revoked) or because no definitive status information is available. Relying parties who require further information SHOULD use the extended response type defined in section 3.2.2.

3.2.2 RTCS extended response

This is an extended response type returning more information than the basic RTCS response:

```
  RtcsResponsesExtended ::= SEQUENCE OF RtcsResponseExtended


  RESPONSEINFO ::= CLASS {
    &status           CertStatus UNIQUE,
    &StatusInfo       OPTIONAL
    } WITH SYNTAX { &status [WITH DETAILS IN &StatusInfo] }


  RtcsResponseExtended ::= SEQUENCE {
    certHash          OtherHash,
    status            RESPONSEINFO.&status({ CertStatus }),
    statusInfo        RESPONSEINFO.&StatusInfo({ CertStatus }{ @status }),
```

```
    attributes   [0] Attributes OPTIONAL
    }


  ResponseTypes RESPONSEINFO ::= {
    { statusOK                                              } |
    { statusNotOK    WITH DETAILS IN InvalidityInfo         } |
    { statusNonAuthoritative
                     WITH DETAILS IN NonAuthoritativeInfo } |
    { statusNoInformation                                   },
    ...
    }


  CertStatus ::= ENUMERATED {
    statusOK (0),
    statusNotOK (1),
    statusNonAuthoritative (2),
    statusNoInformation (3),
    ...
    }
```

In order to provide time information without requiring synchronised clocks (see section 2.7), RTCS uses a relative time value that provides the time as seen by the responder alongside the time at which an event occurred. This eliminates the need for the responder and relying party to have precisely synchronised clocks. The relying party may use the absolute time if they have a mechanism for precise clock synchronisation with the responder, or the difference between the two times to determine how far in the past relative to its own clock the event took place.

```
  RelativeTimeInfo ::= SEQUENCE {
    responderTime    GeneralizedTime,
    eventTime        GeneralizedTime
    }
```

3.2.2.1 Extended status OK

This status value is identical to the basic response equivalent and indicates that the certificate is valid right now.

3.2.2.2 Extended status not-OK

If the certificate has been revoked or rendered invalid in some form, the responder will return a "not-OK" response:

```
InvalidityInfo ::= SEQUENCE {
  invalidityTime   RelativeTimeInfo OPTIONAL,
  invalidityReason CRLReason OPTIONAL
  }
```

invalidityTime indicates the time at which the revocation or invalidation took place, if available.

invalidityReason provides the reason why the certificate was revoked or rendered invalid, if available.

3.2.2.3 Extended status non-authoritative response

This response type may appear when the response is non-authoritative. This situation can occur when the responder being queried obtains information by chaining to another, authoritative responder (an origin server in HTTP terminology) which is temporarily unavailable. Authoritative responders MUST NOT return the statusNonAuthoritative status. Non-authoritative responders may either indicate that no authoritative response is available by omitting the NonAuthoritativeInfo, or provide a non-authoritative response (for example from cached data) in NonAuthoritativeInfo:

```
NonAuthCertStatus ::= CertStatus ( EXCEPT statusNonAuthoritative )

NonAuthoritativeInfo ::= SEQUENCE {
  lastAuthTime     RelativeTime,
  status           RESPONSEINFO.&status({ NonAuthCertStatus }),
  statusInfo       RESPONSEINFO.&StatusInfo({ NonAuthCertStatus }{ @status })
  }
```

lastAuthTime indicates the time at which the last authoritative response was obtained.

The other fields are as defined in section 3.2.2.

3.2.2.4 Extended status no information available

This status value indicates that the queried object doesn't exist, being neither a valid, nor an invalid, certificate (it could for example be a forged

certificate from a third party, or an Excel spreadsheet). Note that this differs from the OCSP "unknown" response, which could mean all manner of things (see section 2.4).

3.2.3 Implementation notes and rationale

The response returned is not intended to be an intrusion into DPD/DPV territory, but simply represents the only response an authenticated dictionary can return. Just as a CRL can only say with certainty "revoked", so an authenticated dictionary can only say with certainty "present" (and conversely, "not present").

The returned status value meets the requirements in section 2.4 (use of an unambiguous status value) and section 2.7 (no reliance on synchronised clocks). The basic response meets the requirements in section 2.6 (use in resource-constrained environments) as well as being the response type of choice in environments where the relying party only cares about a yes/no indicator. This follows the credit card authorisation model, where the merchant only really cares about accepted/declined, and not a 15-page financial statement about why the transaction wasn't accepted. This usage model is exemplified by one commercial S/MIME implementation that boiled the entire certificate checking process down to a single value bCanUseTheDamnThing, because that was the only information that mattered to the user. The response format meets the requirements in section 2.5 (lightweight/realtime operation) since it allows heavyweight signatures or lightweight MACs to be used as required.

For relying parties requiring full information, the extended response provides further details.

A resource-constrained environment may request a basic response and copy the status directly from a fixed location in the response. This makes RTCS amenable for use in crypto tokens or mobile devices that don't have the resources to handle a full implementation. Note however that clients should not assume that response information occurs in the same order as request information when more than one certificate is being queried in an RTCS request. That is, if request information for a certificate is present at position n in the RTCS request then it is not safe to assume that the returned response will similarly contain the certificate status at position n. Instead, clients should use the certificate identifier to match response information to request information.

Responders can be operated in one of two modes. In the most common mode, the responder is authoritative and returns responses directly from the certificate store or a shapshot of the certificate store. In the less common mode, the

responder acts as an access concentrator/transaction coordinator/proxy for other responders. The following discussion of non-authoritative responses and responders borrows from DNS concepts and terminology, which faces a similar situation when handling DNS queries.

When a responder is non-authoritative, it may not be able to return a response to a query directly from the authoritative source, or for performance reasons may return a cached response, just as with DNS and HTTP servers. In this case the responder can return the last authoritative response, along with an indication as to how low ago the response was authoritative. The relying party can then make a decision, based on the age of the cached response and the value of the data involved, to rely on the cached information or to wait for a fresh, authoritative response to become available.

Note that the use of the term "authoritative" differs slightly from its use in DNS. In DNS, cacheing for load-distribution purposes is very common, and mechanisms to handle it are built ito the DNS, whereas with RTCS it would only be used when there isn't a requirement for a hard real-time response. However, RTCS can return an authoritative response (via an access concentrator/transaction coordinator/proxy) without the responder which is being queried itself being authoritative. In HTTP terminology, the source of the authoritative response is an origin server, with caches acting as intermediaries to improve performance. In this case the response is regarded as being authoritative, since it is being forwarded from an authoritative source/origin server. Only a cached response is non-authoritative. This differs from DNS, where the authority of an answer and the authority of a DNS server are synonymous. Further discussion of this style of cacheing model may be found in section 13 of [RFC2616]. This document is recommended reading for anyone considering the use of response cacheing for RTCS performance enhancement purposes.

The HTTP protocol allows the client to override cacheing behaviour on the server through the "Cache-control: no-cache" directive, which indicates that the server must provide an authoritative response. In RTCS this is controlled by the server, with an explicit indication in the response as to whether it is authoritative or not. In other words, the server always provides some form of response, and leaves it to the client to decide whether to utilise it or not. This is based on the view that the client is in a far better position to judge this than the server, since the client/relying party is the one that runs the risk if a decision arising from the validity of the certificate is wrong and not the responder.

[Note: Should add a max-age extension to requests to allow a forced end-to-end reload].

3.2.4 High-speed/High-volume Responder Design

The simple yes/no response option may be used in applications where a high-speed response is required or a high volume of transactions is expected. Observe that the certHash identifier consitutes the application of a high-quality hash function, which should give a perfectly flat distribution of hash values, with all the work being performed by the client. The responder merely has to select n bits of the hash value and perform a lookup in a table of 2^n bits (with appropriate handling of hash chaining/overflows, this is a standard problem from the literature). This means of implementing a certificate status responder is probably the fastest certificate status query mechanism possible.

The use of the CMS format allows further optimisation for high-speed operation, either by taking advantage of hardware acceleration or by using low-overhead MACs instead of high-overhead signatures. PKCS #11 [PKCS11] directly supports the CMS message format, allowing responses to be generated directly by the crypto hardware. Alternatively, CMS supports the use of MACs rather than signatures, allowing responses to be generated with minimal overhead in resource-constrained or high-volume applications.

The use of pre-shared MAC keys presupposes a long-term relationship between an initiator and the RTCS responder, for example where an online transaction processing facility is continuously querying a back-end for certificate status information. However, CMS also allows MAC keys to be established on the fly via standard CMS key exchange mechanisms, which may then be cached by the initiator and RTCS responder for future use. In this manner the initial query necessitates a (relatively) high-overhead private-key operation to unwrap the MAC key (the equivalent of a single signed OCSP response), while subsequent queries can proceed at full speed using MAC'ed messages.

To reduce the load on the server, the MAC key exchange may be initiated by the server rather than the client. In this way the server performs the lightweight public-key wrap while the client has to perform the more heavyweight private-key unwrap.

Further speed optimisations may be obtained by observing that due to the application of the ASN.1 distinguished encoding rules (DER), standard queries and responses have a fixed format, so they may be pre-encoded before transmission and applied as a fixed-format template, and don't need to be decoded on reception because all of the fields are at fixed locations. This means that a high-speed responder can pull the hash value directly from a fixed location in incoming queries, perform the lookup, drop the result into another fixed location in a response template, and enqueue it for transmission back to the initiator.

Various network efficiency considerations need to be taken into account when

implementing this certificate distribution mechanism. For example, a simplistic implementation that performs two writes (the HTTP header and the certificate written seperately) followed by a read will interact badly with TCP delayed-ACK and slow-start. This occurs because the TCP MSS is typically 1460 bytes on a LAN (Ethernet) or 512/536 bytes on a WAN, while HTTP headers are ~200-300 bytes, far less than the MSS. When an HTTP message is first sent, the TCP congestion window begins at one segment, with the TCP slow-start then doubling its size for each ACK. Sending the headers separately will send one short segment and a second MSS-size segment, whereupon the TCP stack will wait for the responder's ACK before continuing. The responder gets both segments, then delays its ACK for 200ms in the hopes of piggybacking it on responder data, which is never sent since it's still waiting for the rest of the HTTP body from the initiator. This behaviour results in a 200ms (+ assorted RTT) delay in each message sent.

There are various other considerations that need to be taken into account in order to provide maximum efficiency. These are covered in depth elsewhere [Spero] [Heidemann] [Nielsen]. In addition, modifications to TCP's behaviour such as the use of 4K initial windows [RFC3390] (designed to reduce small HTTP transfer times to a single RTT) should also ameliorate some of these issues.

A rule of thumb for optimal performance is to combine the HTTP header and data payload into a single write (any reasonable HTTP implementation will do this anyway, thanks to the considerable body of experience that exists for HTTP server performance tuning), and to keep the HTTP headers to a minimum to try and fit data within the TCP MSS. Since this protocol doesn't involve a web browser, there's no need to include the usual headers covering browser versions and languages and so on; a minimal set of content-type/encoding and host and session control information will suffice.

For even better network performance in the presence of large numbers of point queries (single requests rather than an ongoing sequence of requests), RTCS should be run directly over UDP (without any HTTP encapsulation), eliminating the cost of the TCP connection setup and additional protocol overhead. The fixed-format, compact RTCS queries and responses make them ideal for transmission over UDP rather than TCP. Further details on scaling this form of query/response infrastructure may be found in any work that discusses DNS query handling.

4. Security considerations

The legacyID is based on the assumption that DNs in certificates are unique. Although all of X.500 is built upon this assumption, it has been claimed that this may not always be the case. If this is a concern, a DN-based identifier is insufficient to uniquely identify a certificate and the certHash alternative should be used. RTCS always transmits the certHash, so this can

always be relied upon to uniquely identify the certificate even in the presence of duplicate, missing, or arbitrarily broken, DNs.

Appendix A: MIME Wrapping

When RTCS is used in a MIME environment, the S/MIME v3 MIME wrapping rules apply [RFC2633]. The optional smime-type parameter MUST have the value "rtcs-request" for RTCS requests with a file name extension "rrq", and "rtcs-response" for RTCS responses with a file name extension "rrs". All other details are specified in [RFC2633].

Appendix B: Use of RTCS with CMS

This document has described a general-purpose CMS-compatible means of communicating certificate status information. When used in conjunction with other CMS data, it can be inserted as an authenticated or unauthenticated attribute if the CMS data type supports this. The object identifier used to identify RTCS responses communicated as CMS attributes is:

```
rtcsData { 1 3 6 1 4 1 3029 3 1 4 }
```

For example a sender may insert an RTCS response as an authenticated attribute in SignedData to prove that its certificate was valid at the time of signing. An S/MIME gateway or forwarder could insert an RTCS response covering the certificates in the messages it processes as an unauthenticated attribute, allowing clients behind the gateway to determine the validity of the certificates used to sign messages they process without having to perform an RTCS query themselves. This allows RTCS to be used with clients with limited resources, or in situations where only the messaging gateway (but not clients located behind it) have general network access for querying RTCS responders. Note though that in this case the eventual consumer of the information and the RTCS responder require synchronised clocks, since the information is no longer being communicated via an interactive, real-time exchange.

Appendix C: Sample RTCS Messages

<<<Omitted from the draft version to keep the size down>>>

References (Normative)

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels",

Scott Bradner, RFC 2119, March 1997.

[RFC2633] "S/MIME Version 3 Message Specification", RFC 2633, Blake Ramsdell, June 1999.

[RFC2985] "PKCS #9: Selected Object Classes and Attribute Types, Version 2.0", RFC 2985, Magnus Nystrom and Burt Kaliski, November 2000.

[RFC3126] "Electronic Signature Formats for long term electronic signatures", Harri Rasilainen, Denis Pinkas, John Ross, Nick Pope, September 2001.

[RFC3369] "Cryptographic Message Syntax (CMS)", Russ Housley, August 2002.

[RFC3379] "Delegated Path Validation and Delegated Path Discovery Protocol Requirements", Denis Pinkas and Russ Housley, September 2002.

[X.680] "Information Technology - Abstract Syntax Notation One", ITU-T Recommendation X.680 (2002) / ISO/IEC 8824-1:2002, 2002.

References (Informative)

[Gutmann] "Lessons Learned in Implementing and Deploying Crypto Software", Peter Gutmann, Proceedings of the 2002 Usenix Security Symposium, August 2002.

[Heidemann] "Performance Interactions Between P-HTTP and TCP Implementations", J.Heidemann, ACM Computer Communications Review, April 1997.

[Nielsen] "Network Performance Effects of HTTP/1.1, CSS1, and PNG", H.Nielsen, J.Gettys, A.Baird-Smith, E.Prud'hommeaux, H.Wium Lie, and C.Lilley, 24 June 1997, http://www.w3.org/Protocols/HTTP/1.0/Performance/Pipeline.html.

[PKCS11] "PKCS #11 Cryptographic Token Interface Standard, v2.20",

RSA Laboratories, 2003.

[RFC2616] "Hypertext Transfer Protocol, HTTP/1.1", Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee, June 1999.

[RFC3390] "Increasing TCP's Initial Window", RFC 3390, M.Allman, S.Floyd, and C.Partridge, October 2002.

[Spero] "Analysis of HTTP Performance Problems", S.Spero, July 1994, http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html.

Acknowledgements

The author would like to thank Denis Pinkas for providing the motivation to finish this draft, members of the RTCS cabal and users of the cryptlib toolkit for feedback on requirements and comments on issues such as use in constrained environments and handling of superseded certificates, Phil Griffin for ASN.1 technical advice, and an anonymous PKI architect for the observation that "Learning in 80 ms that the cert was good as of a week ago and to not hope for fresher information for another week seems of limited, if any, utility to us or our customers".

Author Address

Peter Gutmann
University of Auckland
Private Bag 92019
Auckland, New Zealand

Email: pgut001@cs.auckland.ac.nz

Full Copyright Statement