X.509 Internet Public Key Infrastructure
Real-time Certificate Status Facility for OCSP - (RTCS)

Status of this Memo

1. Abstract

When the OCSP protocol was defined, the design was based on full compatibility
with CRL-based mechanisms. This requires the use of a complex means of
certificate identification that has resulted in interoperability problems
among implementations, 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).

Fortunately, the authors of the OCSP RFC foresaw this situation by allowing a
client to specify, and a responder to return, more than one type of response.
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. This
allows the responder to provide extended information not possible with CRLs.

In abstract terms, the responder is providing an implementation of an
authenticated dictionary that responds to membership queries from relying

parties. A conventional OCSP responder answers the question "Is x excluded from D?", while an OCSP responder with RTCS capability answers the question "Is x present in D?".

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. The next section defines the protocol based on the design decisions.

2.1 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. Real-world experience has shown that implementors have considerable difficulty with this identifier, leading to interoperability problems among implementations.

A major design goal of RTCS then is to 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).

2.2 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 on any CRL). This problem interacts badly with the one in section 2.1 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 valid 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.

The second major design goal of RTCS then is to 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".

2.3 Use in constrained environments

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 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.

2.4 Reliance on synchronised clocks

OCSP uses timestamps for all responses, assuming that the relying party and responder somehow have perfectly synchronised clocks. This is almost never 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.

3. RTCS

RTCS is designed to provide online, real-time certificate status information by directly reference to a certificate store, in a manner that meets the design goals given in section 2.

3.1 RTCS requests

RTCS makes use of the OCSP AcceptableResponses extension to specify the response types that it will accept. There are two response types, a simple basic status value suitable for use when only a yes/no response is required or in resource-constrained environments, and an extended response type suitable for use when more information is required.

The response types are identified by:

```
   id-pkix-rtcs-basic     OBJECT IDENTIFIER ::= { id-pkix-ocsp TBA }
   id-pkix-rtcs-extended  OBJECT IDENTIFIER ::= { id-pkix-ocsp TBA+1 }
```

In order to identify the certificate, RTCS extends the existing OCSP identifiers to use the following new identifier type:

```
  RtcsIdentifier ::= [2] 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 [RFC 3126], 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 are required. The full definition is given in [RFC 3369].

This identifier is widely used in 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.

Note that the tagging is used to ensure non-interference with existing OCSP
identifiers.

3.1.1 Additional requirements

Since RTCS doesn't depend on synchronised clocks, implementations MUST use the
OCSP Nonce extension to ensure freshness of replies.

3.1.2 Implementation notes and rationale

The certHash identifier meets the requirements in section 2.1 (use of a
widely-accepted, simple, universal identifier for certificates) and section
2.3 (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 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 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.

A resource- or bandwidth-constrained environment may use a pre-generated OCSP
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 OCSP
implementation and that merely populate a pre-generated query with a fresh
nonce and 20-byte certHash.

The full definition of OtherHash, from [RFC 3126], 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 rewriting 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.

[3.2](#) **RTCS response**

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

[Editorial note: These are currently defined in modern ASN.1 for convenience,
 but can be back-ported to ASN.1 '88 (ugh, and with a lot of additional text
 to cover the stuff '88 can't do) later. The following simply say what OCSP
 says with a large amount of text, but in a manner directly usable with an
 ASN.1 compiler, E&OE]

```
    RTCSRESPONSE ::= TYPE-IDENTIFIER

    RtcsResponseBytes ::= SEQUENCE {
        type                 RTCSRESPONSE.&id({ RtcsResponseTypes }),
        response             OCTET STRING (CONTAINING RtcsResponse)
        }

    RtcsResponse ::= SEQUENCE {
        tbsResponseData      RTCSRESPONSE.&Type({ RtcsResponseTypes }{ @.type-
id })
        signatureAlgorithm   AlgorithmIdentifier OPTIONAL,
        signature            BIT STRING OPTIONAL,
        certs                Certificates OPTIONAL
        }

    RtcsResponseTypes RTCSRESPONSE ::= {
        rtscResponseBasic | rtscResponseExtended,
        ...
        }

    rtcsResponseBasic RTSCRESPONSE ::= {
        SYNTAX RtcsResponeBasic ID { id-pkix-rtcs-basic }
        }

    rtcsResponseExtended RTSCRESPONSE ::= {
        SYNTAX RtcsResponeExtended ID { id-pkix-rtcs-extended }
        }

    SignedResponseClass ::= Response
        ( WITH COMPONENTS {
            tbsResponseData( SignedResponseData ) PRESENT,
```

```
          signatureAlgorithm PRESENT,
          signature PRESENT } )

   UnsignedResponseClass ::= Response
      ( WITH COMPONENTS {
          tbsResponseData( UnsignedResponseData ) PRESENT } )

   SignedResponseData ::= ResponseData
      ( WITH COMPONENTS {
          ..., responderID PRESENT } )

   UnsignedResponseData ::= ResponseData
      ( WITH COMPONENTS {
          ..., responderID ABSENT } )
```

3.2.1 RTCS basic response

This is a straightforward yes/no response type:

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

A returned value 'true' indicates that the certificate is valid right now. A
returned value 'false' indicates that the certificate is not 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. Relying parties who require further information
SHOULD use the extended response 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:

```
   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 }),
      extensions    Extensions OPTIONAL
      }

   ResponseTypes RESPONSEINFO ::= {
      { statusOK                                      } |
      { statusRevoked    WITH DETAILS IN RevocationInfo } |
```

```
        { statusSuperseded WITH DETAILS IN SupersededInfo } |
        { statusUnknown                                    },
        ...
        }

    CertStatus ::= ENUMERATED {
        statusOK (0),
        statusRevoked (1),
        statusSuperseded (2),
        statusUnknown (3),
        ...
        }
```

In order to provide time information without requiring synchronised clocks (see section 2.4), 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 revocation 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 revocation took place.

```
    RelativeTimeInfo ::= SEQUENCE {
        localTime          GeneralizedTime,
        timeValue          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/revoked

If the certificate has been revoked or rendered invalid in some form, the responder will return a "revoked" response. Note that the terminology used here is somewhat misleading in that this response corresponds to a "not OK" response, but in X.509 terms this is usually thought of in terms of revocation so this response is named a "revoked" response:

```
    RevocationInfo ::= SEQUENCE {
        revocationTime      RelativeTimeInfo OPTIONAL,
        revocationReason    CRLReason OPTIONAL
        }
```

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

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

3.2.2.3 Extended status superseded

If the certificate has been replace by an updated certificate and the
replacement is available, the responder MAY return the updated certificate
instead of a pure not-OK response:

```
SupersededInfo ::= SEQUENCE {
    replacementTime [0] RelativeTimeInfo OPTIONAL,
    replacementReason   CRLReason OPTIONAL
    replacementCert     Certificate
    }
```

The time and reason have the same meaning as for RevocationInfo.

The replacementCert is the certificate that has superseded the one being
queried.

3.2.2.4 Extended status unknown

This status value indicates that the queried object doesn't exist. Note that
this differs from the standard OCSP "unknown" response, which could mean all
manner of things (see section 2.2).

3.2.3 Implementation notes and rationale

The returned status value meets the requirements in section 2.2 (use of an
unambiguous status value) and section 2.4 (no reliance on synchronised
clocks). The basic response meets the requirements in section 2.3 (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.

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

The superseded response anticipates the request that will immediately follow a
not-OK response to a status query, "What cert should I use instead, then?".
When the RTCS responder is being fed directly from a certificate store, it can
trivially obtain the replacement certificate directly from the store and
return it to the client as an indication of which certificate replaces the one
the client received the not-OK response for.

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 OCSP implementation.

3. 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.

When returning a response, the responder is merely indicating that the queried certificate is currently present in its set of valid certificates. 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 [RFC 3379] and not RTCS.

References

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

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

[RFC 3126] "Electronic Signature Formats for long term electronic signatures",
           Harri Rasilainen, Denis Pinkas, John Ross, Nick Pope, September
           2001.

[RFC 3369] "Cryptographic Message Syntax (CMS)", Russ Housley, August 2002.

[RFC 3379] "Delegated Path Validation and Delegated Path Discovery Protocol
           Requirements", Denis Pinkas and Russ Housley, RFC 3379,
           September 2002.

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

Acknowledgements

The author would like to thank Denis Pinkas for providing the motivation to finish this draft, 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.

Author Address

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

EMail: pgut001@cs.auckland.ac.nz

Full Copyright Statement