Network Working Group
Request for Comments: 5403
Updates: 2203
Category: Standards Track

M. Eisler
NetApp

February 2009

RPCSEC_GSS Version 2

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document describes version 2 of the RPCSEC_GSS protocol. Version 2 is the same as version 1 (specified in RFC 2203) except that support for channel bindings has been added. RPCSEC_GSS allows remote procedure call (RPC) protocols to access the Generic Security Services Application Programming Interface (GSS-API).

Table of Contents

1. Introduction and Motivation

This document describes RPCSEC_GSS version 2 (RPCSEC_GSSv2). RPCSEC_GSSv2 is the same as RPCSEC_GSS version 1 (RPCSEC_GSSv1) [1] except that support for channel bindings [2] has been added. The primary motivation for channel bindings is to securely take advantage of hardware-assisted encryption that might exist at lower levels of the networking protocol stack, such as at the Internet Protocol (IP) layer in the form of IPsec (see [7] and [8] for information on IPsec channel bindings). The secondary motivation is that even if lower levels are not any more efficient at encryption than the RPCSEC_GSS layer, if encryption is occurring at the lower level, it can be redundant at the RPCSEC_GSS level.

RPCSEC_GSSv2 and RPCSEC_GSSv1 are protocols that exchange tokens emitted by the Generic Security Services (GSS) framework, which is defined in [3], and differ only in the support for GSS channel bindings in RPCSEC_GSSv2. GSS itself supports channel bindings, and in theory RPCSEC_GSSv2 could use native GSS channel bindings to achieve the effects described in this section. However, as Section 1.1.6 of [3] states, not all implementations of all GSS mechanisms support channel bindings. This is sufficient justification for the approach taken in this document: modify the RPCSEC_GSS protocol to support channel bindings independent of the capabilities of the GSS mechanism being used.

Once an RPCSEC_GSS target and initiator are mutually assured that they are each using the same secure, end-to-end channel, the overhead of computing message integrity codes (MICs) for authenticating and integrity-protecting RPC requests and replies can be eliminated because the channel is performing the same function. Similarly, if the channel also provides confidentiality, the overhead of RPCSEC_GSS privacy protection can also be eliminated.

The External Data Representation (XDR) [4] description is provided in this document in a way that makes it simple for the reader to extract into a ready-to-compile form. The reader can feed this document into the following shell script to produce the machine-readable XDR description of RPCSEC_GSSv2:

```
<CODE BEGINS>

#!/bin/sh
grep "^  *///" | sed 's?^  */// ??' | sed 's?^  *///$??'

<CODE ENDS>
```

That is, if the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
<CODE BEGINS>

sh extract.sh < spec.txt > rpcsec_gss_v2.x

<CODE ENDS>
```

The effect of the script is to remove leading white space from each line of the specification, plus a sentinel sequence of "///".

1.1. Requirements Language

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 RFC 2119 [5].

2. Channel Bindings Explained

If a channel between two parties is secure, there must be shared information between the two parties. This information might be secret or not. The requirement for secrecy depends on the specifics of the channel.

For example, the shared information could be the concatenation of the public key of the source and destination of the channel (where each public key has a corresponding private key). Suppose the channel is not end-to-end, i.e., a man-in-the-middle (MITM) exists, and there are two channels, one from the initiator to the MITM, and one from the MITM to the target. The MITM cannot simply force each channel to use the same public keys, because a public key derives from a private key, and the key management system for each node will surely assign unique or random private keys. At most, the MITM can force one end of each channel to use the same public key. The MIC of the public keys from the initiator will not be verified by the target, because at least one of the public keys will be different. Similarly, the MIC of the public keys from the target will not be verified by the initiator because at least one of the public keys will be different.

A higher-layer protocol using the secure channel can safely exploit the channel to the mutual benefit of the higher-level parties if each higher-level party can prove:

- They each know the channel's shared information.

- The proof of the knowledge of the shared information is in fact being conveyed by each of the higher-level parties, and not some other entities.

RPCSEC_GSSv2 simply adds an optional round-trip that has the initiator compute a GSS MIC on the channel binding's shared information, and sends the MIC to the target. The target verifies the MIC, and in turn sends its own MIC of the shared information to the initiator that then verifies the target's MIC. This accomplishes three things. First, the initiator and target are mutually authenticated. Second, the initiator and target prove they know the channel's shared information, and thus are using the same channel. Third, the first and second things are done simultaneously.

3. The RPCSEC_GSSv2 Protocol

The RPCSEC_GSSv2 protocol will now be explained. The entire protocol is not presented. Instead the differences between RPCSEC_GSSv2 and RPCSEC_GSSv1 are shown.

3.1. Compatibility with RPCSEC_GSSv1

The functionality of RPCSEC_GSSv1 is fully supported by RPCSEC_GSSv2.

3.2. New Version Number

<CODE BEGINS>

```
/// /*
///  * Copyright (c) 2009 IETF Trust and the persons identified
///  * as the document authors. All rights reserved.
///  *
///  * The document authors are identified in [RFC2203] and
///  * [RFC5403].
///  *
///  * Redistribution and use in source and binary forms, with
///  * or without modification, are permitted provided that the
///  * following conditions are met:
///  *
///  * o Redistributions of source code must retain the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer.
///  *
///  * o Redistributions in binary form must reproduce the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer in the documentation and/or other
///  *   materials provided with the distribution.
///  *
///  * o Neither the name of Internet Society, IETF or IETF
///  *   Trust, nor the names of specific contributors, may be
///  *   used to endorse or promote products derived from this
///  *   software without specific prior written permission.
///  *
///  *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
///  *   AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
///  *   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
///  *   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
///  *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
///  *   EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
///  *   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
///  *   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
///  *   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
///  *   SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
///  *   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
///  *   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
///  *   OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
///  *   IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
///  *   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
///  */
/// /*
///  * This code was derived from [RFC2203]. Please
///  * reproduce this note if possible.
```

```
///  */
///
///  enum rpc_gss_service_t {
///    /* Note: the enumerated value for 0 is reserved. */
///    rpc_gss_svc_none         = 1,
///    rpc_gss_svc_integrity    = 2,
///    rpc_gss_svc_privacy      = 3,
///    rpc_gss_svc_channel_prot = 4 /* new */
///  };
///
///   enum rpc_gss_proc_t {
///     RPCSEC_GSS_DATA          = 0,
///     RPCSEC_GSS_INIT          = 1,
///     RPCSEC_GSS_CONTINUE_INIT = 2,
///     RPCSEC_GSS_DESTROY       = 3,
///     RPCSEC_GSS_BIND_CHANNEL  = 4 /* new */
///  };
///
///  struct rpc_gss_cred_vers_1_t {
///    rpc_gss_proc_t    gss_proc; /* control procedure */
///    unsigned int      seq_num;  /* sequence number */
///    rpc_gss_service_t service;  /* service used */
///    opaque            handle<>; /* context handle */
///  };
///
///  const RPCSEC_GSS_VERS_1 = 1;
///  const RPCSEC_GSS_VERS_2 = 2; /* new */
///
///  union rpc_gss_cred_t switch (unsigned int rgc_version) {
///    case RPCSEC_GSS_VERS_1:
///    case RPCSEC_GSS_VERS_2: /* new */
///      rpc_gss_cred_vers_1_t rgc_cred_v1;
///  };
///

<CODE ENDS>
```

Figure 1

As is apparent from the above, the RPCSEC_GSSv2 credential has the same format as the RPCSEC_GSSv1 credential (albeit corrected so that the definition is in legal XDR description language that is also compatible with [9]; hence, the field "version", a keyword in RFC 1831, is replaced with "rgc_version"). Setting the rgc_version field to 2 indicates that the initiator and target support channel bindings.

3.3. New Procedure - RPCSEC_GSS_BIND_CHANNEL

<CODE BEGINS>

```
///  struct rgss2_bind_chan_MIC_in_args {
///    opaque          rbcmia_bind_chan_hash<>;
///  };
///
///  typedef opaque    rgss2_chan_pref<>;
///  typedef opaque    rgss2_oid<>;
///
///  struct rgss2_bind_chan_verf_args {
///    rgss2_chan_pref rbcva_chan_bind_prefix;
///    rgss2_oid       rbcva_chan_bind_oid_hash;
///    opaque          rbcva_chan_mic<>;
///  };
///
```

<CODE ENDS>

Figure 2

Once an RPCSEC_GSSv2 handle has been established over a secure channel, the initiator MAY issue RPCSEC_GSS_BIND_CHANNEL (Figure 1). Targets MUST support RPCSEC_GSS_BIND_CHANNEL. Like RPCSEC_GSS_INIT and RPCSEC_GSS_CONTINUE_INIT requests, the NULL RPC procedure MUST be used. Unlike those two requests, the arguments of the NULL procedure are not overloaded, because the verifier is of sufficient size for the purpose of RPCSEC_GSS_BIND_CHANNEL. The gss_proc field is set to RPCSEC_GSS_BIND_CHANNEL. The seq_num field is set as if gss_proc were set to RPCSEC_GSS_DATA. The service field is set to rpc_gss_svc_none. The handle field is set to that of an RPCSEC_GSS handle as returned by RPCSEC_GSS_INIT or RPCSEC_GSS_CONTINUE_INIT.

The RPCSEC_GSS_BIND_CHANNEL request is similar to the RPCSEC_GSS_DATA request in that the verifiers of both contain MICs. As described in Section 5.3.1 of [1], when gss_proc is RPCSEC_GSS_DATA, the verifier of an RPC request is set to the output of GSS_GetMIC() on the RPC header. When gss_proc is RPCSEC_GSS_BIND_CHANNEL the verifier of an RPC request is set to the XDR encoding on a value of data type rgss2_bind_chan_verf_args, which includes a MIC as described below. The rgss2_bind_chan_verf_args data type consists of three fields:

- rbcva_chan_bind_prefix. This is the channel binding prefix as described in [2] up to, but excluding, the colon (ASCII 0x3A) that separates the prefix from the suffix.

- o rbcva_chan_bind_hash_oid. This is the object identifier (OID) of the hash algorithm used to compute rbcmia_bind_chan_hash. This field contains an OID encoded in ASN.1 as used by GSS-API in the mech_type argument to GSS_Init_sec_context ([3]). See [6] for the OIDs of the SHA one-way hash algorithms.

- o rbcva_chan_mic. This is the output of GSS_GetMIC() on the concatenation of the XDR-encoded RPC header ("up to and including the credential" as per [1]) and the XDR encoding of an instance of type data rgss2_bind_chan_MIC_in_args. The data type rgss2_bind_chan_MIC_in_args consists of one field, rbcmia_bind_chan_hash, which is a hash of the channel bindings as defined in [2]. The channel bindings are a "canonical octet string encoding of the channel bindings", starting "with the channel bindings prefix followed by a colon (ASCII 0x3A)". The reason a hash of the channel bindings and not the actual channel bindings are used to compute rbcva_chan_mic is that some channel bindings, such as those composed of public keys, can be relatively large, and thus place a higher space burden on the implementations to manage. One way hashes consume less space.

```
<CODE BEGINS>

///  enum rgss2_bind_chan_status {
///    RGSS2_BIND_CHAN_OK           = 0,
///    RGSS2_BIND_CHAN_PREF_NOTSUPP = 1,
///    RGSS2_BIND_CHAN_HASH_NOTSUPP = 2
///  };
///
///  union rgss2_bind_chan_res switch
///     (rgss2_bind_chan_status rbcr_stat) {
///
///    case RGSS2_BIND_CHAN_OK:
///      void;
///
///    case RGSS2_BIND_CHAN_PREF_NOTSUPP:
///      rgss2_chan_pref rbcr_pref_list<>;
///
///    case RGSS2_BIND_CHAN_HASH_NOTSUPP:
///      rgss2_oid       rbcr_oid_list<>;
///  };
///
///  struct rgss2_bind_chan_MIC_in_res {
///    unsigned int        rbcmr_seq_num;
///    opaque              rbcmr_bind_chan_hash<>;
///    rgss2_bind_chan_res rbcmr_res;
///  };
///
```

```
///  struct rgss2_bind_chan_verf_res {
///    rgss2_bind_chan_res rbcvr_res;
///    opaque              rbcvr_mic<>;
///  };
///

<CODE ENDS>
```

Figure 3

The RPCSEC_GSS_BIND_CHANNEL reply is similar to the RPCSEC_GSS_DATA reply in that the verifiers of both contain MICs. When gss_proc is RPCSEC_GSS_DATA, the verifier of an RPC reply is set to the output of GSS_GetMIC() on the seq_num of the credential of the corresponding request (as described in Section 5.3.3.2 of [1]). When gss_proc is RPCSEC_GSS_BIND_CHANNEL, the verifier of an RPC reply is set to the XDR encoding of an instance of data type rgss2_bind_chan_verf_res, which includes a MIC as described below. The data type rgss2_bind_chan_verf_res consists of two fields.

- o rbcvr_res. The data type of this field is rgss2_bind_chan_res. The rgss2_bind_chan_res data type is a switched union consisting of three cases switched on the status contained in the rbcr_stat field.

  * RGSS2_BIND_CHAN_OK. If this status is returned, the target accepted the channel bindings, and successfully verified rbcva_chan_mic in the request. No additional results will be in rbcvr_res.

  * RGSS2_BIND_CHAN_PREF_NOTSUPP. If this status is returned, the target did not support the prefix in the rbcva_chan_bind_prefix field of the arguments, and thus the RPCSEC_GSS_BIND_CHANNEL request was rejected. The target returned a list of prefixes it does support in the field rbcr_pref_list. Note that a channel can have multiple channel bindings each with different prefixes. The initiator is free to pick its preferred prefix. If the target does not support the prefix, the status RGSS2_BIND_CHAN_PREF_NOTSUPP will be returned, and the initiator can select its next most preferred prefix among the prefixes the target does support.

  * RGSS2_BIND_CHAN_HASH_NOTSUPP. If this status is returned, the target did not support the hash algorithm identified in the rbcva_chan_bind_hash_oid field of the arguments, and thus the RPCSEC_GSS_BIND_CHANNEL request was rejected. The target

returned a list of OIDs of hash algorithms it does support in the field rbcr_oid_list. The array rbcr_oid_list MUST have one or more elements.

- rbcvr_mic. The value of this field is equal to the output of GSS_GetMIC() on the XDR encoding of an instance of data type rgss2_bind_chan_MIC_in_res. The data type rgss2_bind_chan_MIC_in_res consists of three fields.

  * rbcmr_seq_num. The value of this field is equal to the field seq_num in the RPCSEC_GSS credential (data type rpc_gss_cred_vers_1_t).

  * rbcmr_bind_chan_hash. This is the result of the one way hash of the channel bindings (including the prefix). If rbcr_stat is not RGSS2_BIND_CHAN_HASH_NOTSUPP, then the hash algorithm that is used to compute rbcmr_bind_chan_hash is that identified by the rbcva_chan_bind_oid_hash field in the arguments to RPCSEC_GSS_BIND_CHANNEL. If rbcr_stat is RGSS2_BIND_CHAN_HASH_NOTSUPP, then the hash algorithm used to compute rbcmr_bind_chan_hash is that identified by rbcr_oid_list[0] in the results.

  * rbcmr_res. The value of this field is equal to the value of the rbcvr_res field.

3.4. New Security Service - rpc_gss_svc_channel_prot

RPCSEC_GSSv2 targets MUST support rpc_gss_svc_channel_prot.

The rpc_gss_svc_channel_prot service (Figure 1) is valid only if RPCSEC_GSSv2 is being used, an RPCSEC_GSS_BIND_CHANNEL procedure has been executed successfully, and the secure channel still exists. When rpc_gss_svc_channel_prot is used, the RPC requests and replies are similar to those of rpc_gss_svc_none except that the verifiers on the request and reply always have the flavor set to AUTH_NONE, and the contents are zero length.

Note that even though NULL verifiers are used when rpc_gss_svc_channel_prot is used, non-NULL RPCSEC_GSS credentials are used. In order to identify the principal sending the request, the same credential is used as before, except that service field is set to rpc_gss_svc_channel_prot.

4. Version Negotiation

An initiator that supports version 2 of RPCSEC_GSS simply issues an RPCSEC_GSS request with the rgc_version field set to RPCSEC_GSS_VERS_2. If the target does not recognize RPCSEC_GSS_VERS_2, the target will return an RPC error per Section 5.1 of [1].

The initiator MUST NOT attempt to use an RPCSEC_GSS handle returned by version 2 of a target with version 1 of the same target. The initiator MUST NOT attempt to use an RPCSEC_GSS handle returned by version 1 of a target with version 2 of the same target.

5. Native GSS Channel Bindings

To ensure interoperability, implementations of RPCSEC_GSSv2 SHOULD NOT transfer tokens between the initiator and target that use native GSS channel bindings (as defined in Section 1.1.6 of [3]).

6. Operational Recommendation for Deployment

RPCSEC_GSSv2 is a superset of RPCSEC_GSSv1, and so can be used in all situations where RPCSEC_GSSv1 is used. RPCSEC_GSSv2 should be used when the new functionality, channel bindings, is desired or needed.

7. Implementation Notes

Once a successful RPCSEC_GSS_BIND_CHANNEL procedure has been performed on an RPCSEC_GSSv2 context handle, the initiator's implementation may map application requests for rpc_gss_svc_none and rpc_gss_svc_integrity to rpc_gss_svc_channel_prot credentials. And if the secure channel has privacy enabled, requests for rpc_gss_svc_privacy can also be mapped to rpc_gss_svc_channel_prot.

8. Acknowledgments

Nicolas Williams had the idea for extending RPCSEC_GSS to support channel bindings. Alex Burlyga, Lars Eggert, Pasi Eronen, and Dan Romascanu reviewed the document and gave valuable feedback for improving its readability.

9. Security Considerations

The base security considerations consist of:

- All security considerations from [1].

- All security considerations from [2].

- o All security considerations from the actual secure channel being used.

Even though RPCSEC_GSS_DATA requests that use rpc_gss_svc_channel_prot protection do not involve construction of more GSS tokens, the target SHOULD stop allowing RPCSEC_GSS_DATA requests with rpc_gss_svc_channel_prot protection once the GSS context expires.

With the use of channel bindings, it becomes extremely critical that the message integrity code (MIC) used by the GSS mechanism that RPCSEC_GSS is using be difficult to forge. While this requirement is true for RPCSEC_GSSv1, and indeed any protocol that uses GSS MICs, the distinction in the seriousness is that for RPCSEC_GSSv1, forging a single MIC at most allows the attacker to succeed in injecting one bogus request. Whereas, with RPCSEC_GSSv2 combined with channel bindings, by forging a single MIC the attacker will succeed in injecting bogus requests as long as the channel exists. An example illustrates. Suppose we have an RPCSEC_GSSv1 initiator, a man-in-the-middle (MITM), an RPCSEC_GSSv1 target, and an RPCSEC_GSSv2 target. The attack is as follows.

- o The MITM intercepts the initiator’s RPCSEC_GSSv1 RPCSEC_GSS_INIT message and changes the version number from 1 to 2 before forwarding to the RPCSEC_GSSv2 target, and changes the reply’s version number from 2 to 1 before forwarding to the RPCSEC_GSSv1 initiator. Neither the client nor the server notice.

- o Once the RPCSEC_GSS handle is in an established state, the initiator sends its first RPCSEC_GSS_DATA request. The MITM constructs an RPCSEC_GSS_BIND_CHANNEL request, using the message integrity code (MIC) of the RPCSEC_GSS_DATA request. It is likely the RPCSEC_GSSv2 target will reject the request. The MITM continues to reiterate each time the initiator sends another RPCSEC_GSS_DATA request. With enough iterations, the probability of a MIC from an RPCSEC_GSS_DATA being successfully verified in the forged RPCSEC_GSS_BIND_CHANNEL increases. Once the MITM succeeds, it can send RPCSEC_GSS_DATA requests with a security service of rpc_gss_svc_channel_prot, which does not have MICs in the RPC request’s verifier.

The implementation of RPCSEC_GSSv2 can use at least two methods to thwart these attacks.

- o The target SHOULD require a stronger MIC when sending an RPCSEC_GSS_BIND_CHANNEL request instead of an RPCSEC_GSS_DATA request -- e.g., if HMACs are used for the MICs, require the widest possible HMAC (in terms of bit length) that the GSS mechanism supports. If HMACs are being used, and the target expects N RPCSEC_GSS_DATA requests to be sent on the context before it expires, then the target SHOULD require an HMAC for RPCSEC_GSS_BIND_CHANNEL that is log base 2 N bits longer than what it normally requires for RPCSEC_GSS_DATA requests. If a long enough MIC is not available, then the target could artificially limit the number of RPCSEC_GSS_DATA requests it will allow on the context before deleting the context.

- Each time an RPCSEC_GSSv2 target experiences a failure to verify the MIC of an RPCSEC_GSS_BIND_CHANNEL request, it SHOULD reduce the lifetime of the underlying GSS context, by a significant fraction, thereby preventing the MITM from using the established context for its attack. A possible heuristic is that if the target believes the possibility that failure to verify the MIC was because of an attack is X percent, then the context's lifetime would be reduced by X percent. For simplicity, an implementer might set X to be 50 percent, so that the context lifetime is halved on each failed verification of an RPCSEC_GSS_BIND_CHANNEL request and thus rapidly reduced to zero on subsequent requests. For example, with a context lifetime of 8 hours (or 28800 seconds), 15 failed attempts by the MITM would cause the context to be destroyed.

A method of mitigation that was considered was to protect the RPCSEC_GSS version number with RPCSEC_GSSv2's RPCSEC_GSS_INIT and RPCSEC_GSS_CONTINUE_INIT tokens. Thus, the version number of RPCSEC_GSS would be in the tokens. This method does not completely mitigate the attack; it just moves the MIC guessing to the RPCSEC_GSS_INIT message. In addition, without changing GSS, or the GSS mechanism, there is no way to include the RPCSEC_GSS version number in the tokens. So for these reasons this method was not selected.

10. References

10.1. Normative References

[1] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, September 1997.

[2] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[3] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[4] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

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

[6] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

10.2. Informative References

[7] Williams, N., "IPsec Channels: Connection Latching", Work in Progress, November 2008.

[8] Williams, N., "End-Point Channel Bindings for IPsec Using IKEv2 and Public Keys", Work in Progress, April 2008.

[9] Srinivasan, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 1831, August 1995.

Author's Address

Mike Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
US

Phone: +1-719-599-9026
EMail: mike@eisler.com