Network Working Group
Request for Comments: 3645
Updates: 2845
Category: Standards Track
S. Kwan
P. Garg
J. Gilroy
L. Esibov
J. Westhead
Microsoft Corp.
R. Hall
Lucent Technologies
October 2003

Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)

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

Abstract

The Secret Key Transaction Authentication for DNS (TSIG) protocol provides transaction level authentication for DNS. TSIG is extensible through the definition of new algorithms. This document specifies an algorithm based on the Generic Security Service Application Program Interface (GSS-API) (RFC2743). This document updates RFC 2845.

Table of Contents

1. Introduction

The Secret Key Transaction Authentication for DNS (TSIG) [RFC2845] protocol was developed to provide a lightweight authentication and integrity of messages between two DNS entities, such as client and server or server and server. TSIG can be used to protect dynamic update messages, authenticate regular message or to off-load complicated DNSSEC [RFC2535] processing from a client to a server and still allow the client to be assured of the integrity of the answers.

The TSIG protocol [RFC2845] is extensible through the definition of new algorithms. This document specifies an algorithm based on the Generic Security Service Application Program Interface (GSS-API) [RFC2743]. GSS-API is a framework that provides an abstraction of security to the application protocol developer. The security services offered can include authentication, integrity, and confidentiality.

The GSS-API framework has several benefits:

* Mechanism and protocol independence. The underlying mechanisms that realize the security services can be negotiated on the fly and varied over time. For example, a client and server MAY use Kerberos [RFC1964] for one transaction, whereas that same server MAY use SPKM [RFC2025] with a different client.

* The protocol developer is removed from the responsibility of creating and managing a security infrastructure. For example, the developer does not need to create new key distribution or key management systems. Instead the developer relies on the security service mechanism to manage this on its behalf.

The scope of this document is limited to the description of an authentication mechanism only. It does not discuss and/or propose an authorization mechanism. Readers that are unfamiliar with GSS-API concepts are encouraged to read the characteristics and concepts section of [RFC2743] before examining this protocol in detail. It is also assumed that the reader is familiar with [RFC2845], [RFC2930], [RFC1034] and [RFC1035].

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Algorithm Overview

In GSS, client and server interact to create a "security context". The security context can be used to create and verify transaction signatures on messages between the two parties. A unique security context is required for each unique connection between client and server.

Creating a security context involves a negotiation between client and server. Once a context has been established, it has a finite lifetime for which it can be used to secure messages. Thus there are three states of a context associated with a connection:

```
                  +----------+
                  |          |
                  V          |
          +---------------+  |
          | Uninitialized |  |
          |               |  |
          +---------------+  |
                  |          |
                  V          |
          +---------------+  |
          | Negotiating   |  |
          | Context       |  |
          +---------------+  |
                  |          |
                  V          |
          +---------------+  |
          | Context       |  |
          | Established   |  |
          +---------------+  |
                  |          |
                  +----------+
```

Every connection begins in the uninitialized state.

2.1. GSS Details

Client and server MUST be locally authenticated and have acquired default credentials before using this protocol as specified in Section 1.1.1 "Credentials" in RFC 2743 [RFC2743].

The GSS-TSIG algorithm consists of two stages:

I. Establish security context. The Client and Server use the GSS_Init_sec_context and GSS_Accept_sec_context APIs to generate the tokens that they pass to each other using [RFC2930] as a transport mechanism.

II. Once the security context is established it is used to generate and verify signatures using GSS_GetMIC and GSS_VerifyMIC APIs. These signatures are exchanged by the Client and Server as a part of the TSIG records exchanged in DNS messages sent between the Client and Server, as described in [RFC2845].

2.2. Modifications to the TSIG protocol (RFC 2845)

Modification to RFC 2845 allows use of TSIG through signing server's response in an explicitly specified place in multi message exchange between two DNS entities even if client's request wasn't signed.

Specifically, Section 4.2 of RFC 2845 MUST be modified as follows:

Replace:
"The server MUST not generate a signed response to an unsigned request."

With:
"The server MUST not generate a signed response to an unsigned request, except in case of response to client’s unsigned TKEY query if secret key is established on server side after server processed client’s query. Signing responses to unsigned TKEY queries MUST be explicitly specified in the description of an individual secret key establishment algorithm."

3. Client Protocol Details

A unique context is required for each server to which the client sends secure messages. A context is identified by a context handle. A client maintains a mapping of servers to handles:

(target_name, key_name, context_handle)

The value key_name also identifies a context handle. The key_name is the owner name of the TKEY and TSIG records sent between a client and a server to indicate to each other which context MUST be used to process the current request.

DNS client and server MAY use various underlying security mechanisms to establish security context as described in sections 3 and 4. At the same time, in order to guarantee interoperability between DNS clients and servers that support GSS-TSIG it is REQUIRED that security mechanism used by client enables use of Kerberos v5 (see Section 9 for more information).

3.1. Negotiating Context

In GSS, establishing a security context involves the passing of opaque tokens between the client and the server. The client generates the initial token and sends it to the server. The server processes the token and if necessary, returns a subsequent token to the client. The client processes this token, and so on, until the negotiation is complete. The number of times the client and server exchange tokens depends on the underlying security mechanism. A completed negotiation results in a context handle.

The TKEY resource record [RFC2930] is used as the vehicle to transfer tokens between client and server. The TKEY record is a general mechanism for establishing secret keys for use with TSIG. For more information, see [RFC2930].

3.1.1. Call GSS_Init_sec_context

To obtain the first token to be sent to a server, a client MUST call GSS_Init_sec_context API.

The following input parameters MUST be used. The outcome of the call is indicated with the output values below. Consult Sections 2.2.1, "GSS_Init_sec_context call", of [RFC2743] for syntax definitions.

```
INPUTS
  CREDENTIAL HANDLE claimant_cred_handle = NULL (NULL specifies "use
      default").  Client MAY instead specify some other valid
      handle to its credentials.
  CONTEXT HANDLE input_context_handle  = 0
  INTERNAL NAME  targ_name             = "DNS@<target_server_name>"
  OBJECT IDENTIFIER mech_type          = Underlying security
      mechanism chosen by implementers.  To guarantee
      interoperability of the implementations of the GSS-TSIG
      mechanism client MUST specify a valid underlying security
      mechanism that enables use of Kerberos v5 (see Section 9 for
      more information).
  OCTET STRING   input_token           = NULL
  BOOLEAN        replay_det_req_flag   = TRUE
  BOOLEAN        mutual_req_flag       = TRUE
  BOOLEAN        deleg_req_flag        = TRUE
  BOOLEAN        sequence_req_flag     = TRUE
  BOOLEAN        anon_req_flag         = FALSE
  BOOLEAN        integ_req_flag        = TRUE
  INTEGER        lifetime_req          = 0 (0 requests a default
      value).  Client MAY instead specify another upper bound for the
      lifetime of the context to be established in seconds.
  OCTET STRING   chan_bindings         = Any valid channel bindings
      as specified in Section 1.1.6 "Channel Bindings" in [RFC2743]
```

```
OUTPUTS
  INTEGER        major_status
  CONTEXT HANDLE output_context_handle
  OCTET STRING   output_token
  BOOLEAN        replay_det_state
  BOOLEAN        mutual_state
  INTEGER        minor_status
  OBJECT IDENTIFIER mech_type
  BOOLEAN        deleg_state
```

```
BOOLEAN        sequence_state
BOOLEAN        anon_state
BOOLEAN        trans_state
BOOLEAN        prot_ready_state
BOOLEAN        conf_avail
BOOLEAN        integ_avail
INTEGER        lifetime_rec
```

If returned major_status is set to one of the following errors:

```
GSS_S_DEFECTIVE_TOKEN
GSS_S_DEFECTIVE_CREDENTIAL
GSS_S_BAD_SIG (GSS_S_BAD_MIC)
GSS_S_NO_CRED
GSS_S_CREDENTIALS_EXPIRED
GSS_S_BAD_BINDINGS
GSS_S_OLD_TOKEN
GSS_S_DUPLICATE_TOKEN
GSS_S_NO_CONTEXT
GSS_S_BAD_NAMETYPE
GSS_S_BAD_NAME
GSS_S_BAD_MECH
GSS_S_FAILURE
```

then the client MUST abandon the algorithm and MUST NOT use the GSS-TSIG algorithm to establish this security context. This document does not prescribe which other mechanism could be used to establish a security context. Next time when this client needs to establish security context, the client MAY use GSS-TSIG algorithm.

Success values of major_status are GSS_S_CONTINUE_NEEDED and GSS_S_COMPLETE. The exact success code is important during later processing.

The values of replay_det_state and mutual_state indicate if the security package provides replay detection and mutual authentication, respectively. If returned major_status is GSS_S_COMPLETE AND one or both of these values are FALSE, the client MUST abandon this algorithm.

Client's behavior MAY depend on other OUTPUT parameters according to the policy local to the client.

The handle output_context_handle is unique to this negotiation and is stored in the client's mapping table as the context_handle that maps to target_name.

3.1.2. Send TKEY Query to Server

An opaque output_token returned by GSS_Init_sec_context is transmitted to the server in a query request with QTYPE=TKEY. The token itself will be placed in a Key Data field of the RDATA field in the TKEY resource record in the additional records section of the query. The owner name of the TKEY resource record set queried for and the owner name of the supplied TKEY resource record in the additional records section MUST be the same. This name uniquely identifies the security context to both the client and server, and thus the client SHOULD use a value which is globally unique as described in [RFC2930]. To achieve global uniqueness, the name MAY contain a UUID/GUID [ISO11578].

```
TKEY Record
  NAME = client-generated globally unique domain name string
         (as described in [RFC2930])
  RDATA
     Algorithm Name      = gss-tsig
     Mode                = 3 (GSS-API negotiation - per [RFC2930])
     Key Size            = size of output_token in octets
     Key Data            = output_token
```

The remaining fields in the TKEY RDATA, i.e., Inception, Expiration, Error, Other Size and Data Fields, MUST be set according to [RFC2930].

The query is transmitted to the server.

Note: if the original client call to GSS_Init_sec_context returned any major_status other than GSS_S_CONTINUE_NEEDED or GSS_S_COMPLETE, then the client MUST NOT send TKEY query. Client's behavior in this case is described above in Section 3.1.1.

3.1.3. Receive TKEY Query-Response from Server

Upon the reception of the TKEY query the DNS server MUST respond according to the description in Section 4. This section specifies the behavior of the client after it receives the matching response to its query.

The next processing step depends on the value of major_status from the most recent call that client performed to GSS_Init_sec_context: either GSS_S_COMPLETE or GSS_S_CONTINUE.

3.1.3.1. Value of major_status == GSS_S_COMPLETE

If the last call to GSS_Init_sec_context yielded a major_status value of GSS_S_COMPLETE and a non-NULL output_token was sent to the server, then the client side component of the negotiation is complete and the client is awaiting confirmation from the server.

Confirmation is in the form of a query response with RCODE=NOERROR and with the last client supplied TKEY record in the answer section of the query. The response MUST be signed with a TSIG record. Note that the server is allowed to sign a response to unsigned client's query due to modification to the RFC 2845 specified in Section 2.2 above. The signature in the TSIG record MUST be verified using the procedure detailed in section 5, Sending and Verifying Signed Messages. If the response is not signed, OR if the response is signed but the signature is invalid, then an attacker has tampered with the message in transit or has attempted to send the client a false response. In this case, the client MAY continue waiting for a response to its last TKEY query until the time period since the client sent last TKEY query expires. Such a time period is specified by the policy local to the client. This is a new option that allows the DNS client to accept multiple answers for one query ID and select one (not necessarily the first one) based on some criteria.

If the signature is verified, the context state is advanced to Context Established. Proceed to section 3.2 for usage of the security context.

3.1.3.2. Value of major_status == GSS_S_CONTINUE_NEEDED

If the last call to GSS_Init_sec_context yielded a major_status value of GSS_S_CONTINUE_NEEDED, then the negotiation is not yet complete. The server will return to the client a query response with a TKEY record in the Answer section. If the DNS message error is not NO_ERROR or error field in the TKEY record is not 0 (i.e., no error), then the client MUST abandon this negotiation sequence. The client MUST delete an active context by calling GSS_Delete_sec_context providing the associated context_handle. The client MAY repeat the negotiation sequence starting with the uninitialized state as described in section 3.1. To prevent infinite looping the number of attempts to establish a security context MUST be limited to ten or less.

If the DNS message error is NO_ERROR and the error field in the TKEY record is 0 (i.e., no error), then the client MUST pass a token specified in the Key Data field in the TKEY resource record to

GSS_Init_sec_context using the same parameters values as in previous call except values for CONTEXT HANDLE input_context_handle and OCTET STRING input_token as described below:

```
INPUTS
  CONTEXT HANDLE input_context_handle  = context_handle (this is the
       context_handle corresponding to the key_name which is the
       owner name of the TKEY record in the answer section in the
       TKEY query response)

  OCTET STRING   input_token           = token from Key field of
                                         TKEY record
```

Depending on the following OUTPUT values of GSS_Init_sec_context

```
     INTEGER        major_status
     OCTET STRING   output_token
```

the client MUST take one of the following actions:

If OUTPUT major_status is set to one of the following values:

```
     GSS_S_DEFECTIVE_TOKEN
     GSS_S_DEFECTIVE_CREDENTIAL
     GSS_S_BAD_SIG (GSS_S_BAD_MIC)
     GSS_S_NO_CRED
     GSS_S_CREDENTIALS_EXPIRED
     GSS_S_BAD_BINDINGS
     GSS_S_OLD_TOKEN
     GSS_S_DUPLICATE_TOKEN
     GSS_S_NO_CONTEXT
     GSS_S_BAD_NAMETYPE
     GSS_S_BAD_NAME
     GSS_S_BAD_MECH
     GSS_S_FAILURE
```

the client MUST abandon this negotiation sequence. This means that the client MUST delete an active context by calling GSS_Delete_sec_context providing the associated context_handle. The client MAY repeat the negotiation sequence starting with the uninitialized state as described in section 3.1. To prevent infinite looping the number of attempts to establish a security context MUST be limited to ten or less.

If OUTPUT major_status is GSS_S_CONTINUE_NEEDED OR GSS_S_COMPLETE then client MUST act as described below.

If the response from the server was signed, and the OUTPUT major_status is GSS_S_COMPLETE,then the signature in the TSIG record MUST be verified using the procedure detailed in section 5, Sending and Verifying Signed Messages. If the signature is invalid, then the client MUST abandon this negotiation sequence. This means that the client MUST delete an active context by calling GSS_Delete_sec_context providing the associated context_handle. The client MAY repeat the negotiation sequence starting with the uninitialized state as described in section 3.1. To prevent infinite looping the number of attempts to establish a security context MUST be limited to ten or less.

If major_status is GSS_S_CONTINUE_NEEDED the negotiation is not yet finished. The token output_token MUST be passed to the server in a TKEY record by repeating the negotiation sequence beginning with section 3.1.2. The client MUST place a limit on the number of continuations in a context negotiation to prevent endless looping. Such limit SHOULD NOT exceed value of 10.

If major_status is GSS_S_COMPLETE and output_token is non-NULL, the client-side component of the negotiation is complete but the token output_token MUST be passed to the server by repeating the negotiation sequence beginning with section 3.1.2.

If major_status is GSS_S_COMPLETE and output_token is NULL, context negotiation is complete. The context state is advanced to Context Established. Proceed to section 3.2 for usage of the security context.

3.2. Context Established

When context negotiation is complete, the handle context_handle MUST be used for the generation and verification of transaction signatures.

The procedures for sending and receiving signed messages are described in section 5, Sending and Verifying Signed Messages.

3.2.1. Terminating a Context

When the client is not intended to continue using the established security context, the client SHOULD delete an active context by calling GSS_Delete_sec_context providing the associated context_handle, AND client SHOULD delete the established context on the DNS server by using TKEY RR with the Mode field set to 5, i.e., "key deletion" [RFC2930].

4. Server Protocol Details

As on the client-side, the result of a successful context negotiation is a context handle used in future generation and verification of the transaction signatures.

A server MAY be managing several contexts with several clients. Clients identify their contexts by providing a key name in their request. The server maintains a mapping of key names to handles:

(key_name, context_handle)

4.1. Negotiating Context

A server MUST recognize TKEY queries as security context negotiation messages.

4.1.1. Receive TKEY Query from Client

Upon receiving a query with QTYPE = TKEY, the server MUST examine whether the Mode and Algorithm Name fields of the TKEY record in the additional records section of the message contain values of 3 and gss-tsig, respectively. If they do, then the (key_name, context_handle) mapping table is searched for the key_name matching the owner name of the TKEY record in the additional records section of the query. If the name is found in the table and the security context for this name is established and not expired, then the server MUST respond to the query with BADNAME error in the TKEY error field. If the name is found in the table and the security context is not established, the corresponding context_handle is used in subsequent GSS operations. If the name is found but the security context is expired, then the server deletes this security context, as described in Section 4.2.1, and interprets this query as a start of new security context negotiation and performs operations described in Section 4.1.2 and 4.1.3. If the name is not found, then the server interprets this query as a start of new security context negotiation and performs operations described in Section 4.1.2 and 4.1.3.

4.1.2. Call GSS_Accept_sec_context

The server performs its side of a context negotiation by calling GSS_Accept_sec_context. The following input parameters MUST be used. The outcome of the call is indicated with the output values below. Consult Sections 2.2.2 "GSS_Accept_sec_context call" of the RFC 2743 [RFC2743] for syntax definitions.

```
   INPUTS
     CONTEXT HANDLE input_context_handle  = 0 if new negotiation,
                                             context_handle matching
                                          key_name if ongoing negotiation
     OCTET STRING   input_token           = token specified in the Key
           field from TKEY RR (from Additional records Section of
           the client's query)

     CREDENTIAL HANDLE acceptor_cred_handle = NULL (NULL specifies "use
           default").  Server MAY instead specify some other valid
           handle to its credentials.
     OCTET STRING   chan_bindings          = Any valid channel bindings
           as specified in Section 1.1.6 "Channel Bindings" in [RFC2743]

   OUTPUTS
     INTEGER        major_status
     CONTEXT_HANDLE output_context_handle
     OCTET STRING   output_token
     INTEGER        minor_status
     INTERNAL NAME  src_name
     OBJECT IDENTIFIER  mech_type
     BOOLEAN        deleg_state
     BOOLEAN        mutual_state
     BOOLEAN        replay_det_state
     BOOLEAN        sequence_state
     BOOLEAN        anon_state
     BOOLEAN        trans_state
     BOOLEAN        prot_ready_state
     BOOLEAN        conf_avail
     BOOLEAN        integ_avail
     INTEGER        lifetime_rec
     CONTEXT_HANDLE delegated_cred_handle
```

If this is the first call to GSS_Accept_sec_context in a new negotiation, then output_context_handle is stored in the server's key-mapping table as the context_handle that maps to the name of the TKEY record.

4.1.3. Send TKEY Query-Response to Client

The server MUST respond to the client with a TKEY query response with RCODE = NOERROR, that contains a TKEY record in the answer section.

If OUTPUT major_status is one of the following errors the error field in the TKEY record set to BADKEY.

GSS_S_DEFECTIVE_TOKEN
GSS_S_DEFECTIVE_CREDENTIAL
GSS_S_BAD_SIG (GSS_S_BAD_MIC)
GSS_S_DUPLICATE_TOKEN
GSS_S_OLD_TOKEN
GSS_S_NO_CRED
GSS_S_CREDENTIALS_EXPIRED
GSS_S_BAD_BINDINGS
GSS_S_NO_CONTEXT
GSS_S_BAD_MECH
GSS_S_FAILURE

If OUTPUT major_status is set to GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED then server MUST act as described below.

If major_status is GSS_S_COMPLETE the server component of the negotiation is finished. If output_token is non-NULL, then it MUST be returned to the client in a Key Data field of the RDATA in TKEY. The error field in the TKEY record is set to NOERROR. The message MUST be signed with a TSIG record as described in section 5, Sending and Verifying Signed Messages. Note that server is allowed to sign a response to unsigned client's query due to modification to the RFC 2845 specified in Section 2.2 above. The context state is advanced to Context Established. Section 4.2 discusses the usage of the security context.

If major_status is GSS_S_COMPLETE and output_token is NULL, then the TKEY record received from the client MUST be returned in the Answer section of the response. The message MUST be signed with a TSIG record as described in section 5, Sending and Verifying Signed Messages. Note that server is allowed to sign a response to unsigned client's query due to modification to the RFC 2845 specified in section 2.2 above. The context state is advanced to Context Established. Section 4.2 discusses the usage of the security context.

If major_status is GSS_S_CONTINUE_NEEDED, the server component of the negotiation is not yet finished. The server responds to the TKEY query with a standard query response, placing in the answer section a TKEY record containing output_token in the Key Data RDATA field. The error field in the TKEY record is set to NOERROR. The server MUST limit the number of times that a given context is allowed to repeat, to prevent endless looping. Such limit SHOULD NOT exceed value of 10.

In all cases, except if major_status is GSS_S_COMPLETE and output_token is NULL, other TKEY record fields MUST contain the following values:

```
NAME = key_name
RDATA
   Algorithm Name      = gss-tsig
   Mode                = 3 (GSS-API negotiation - per [RFC2930])
   Key Size            = size of output_token in octets
```

The remaining fields in the TKEY RDATA, i.e., Inception, Expiration, Error, Other Size and Data Fields, MUST be set according to [RFC2930].

4.2. Context Established

When context negotiation is complete, the handle context_handle is used for the generation and verification of transaction signatures. The handle is valid for a finite amount of time determined by the underlying security mechanism. A server MAY unilaterally terminate a context at any time (see section 4.2.1).

Server SHOULD limit the amount of memory used to cache established contexts.

The procedures for sending and receiving signed messages are given in section 5, Sending and Verifying Signed Messages.

4.2.1. Terminating a Context

A server can terminate any established context at any time. The server MAY hint to the client that the context is being deleted by including a TKEY RR in a response with the Mode field set to 5, i.e., "key deletion" [RFC2930]. An active context is deleted by calling GSS_Delete_sec_context providing the associated context_handle.

5. Sending and Verifying Signed Messages

5.1. Sending a Signed Message - Call GSS_GetMIC

The procedure for sending a signature-protected message is specified in [RFC2845]. The data to be passed to the signature routine includes the whole DNS message with specific TSIG variables appended. For the exact format, see [RFC2845]. For this protocol, use the following TSIG variable values:

```
      TSIG Record
        NAME = key_name that identifies this context
        RDATA
           Algorithm Name = gss-tsig
```

Assign the remaining fields in the TSIG RDATA appropriate values as described in [RFC2845].

The signature is generated by calling GSS_GetMIC. The following input parameters MUST be used. The outcome of the call is indicated with the output values specified below. Consult Sections 2.3.1 "GSS_GetMIC call" of the RFC 2743[RFC2743] for syntax definitions.

```
   INPUTS
     CONTEXT HANDLE context_handle = context_handle for key_name
     OCTET STRING   message        = outgoing message plus TSIG
                                     variables (per [RFC2845])
     INTEGER qop_req               = 0 (0 requests a default
         value).  Caller MAY instead specify other valid value (for
         details see Section 1.2.4 in [RFC2743])

   OUTPUTS
     INTEGER        major_status
     INTEGER        minor_status
     OCTET STRING   per_msg_token
```

If major_status is GSS_S_COMPLETE, then signature generation succeeded. The signature in per_msg_token is inserted into the Signature field of the TSIG RR and the message is transmitted.

If major_status is GSS_S_CONTEXT_EXPIRED, GSS_S_CREDENTIALS_EXPIRED or GSS_S_FAILURE the caller MUST delete the security context, return to the uninitialized state and SHOULD negotiate a new security context, as described above in Section 3.1

If major_status is GSS_S_NO_CONTEXT, the caller MUST remove the entry for key_name from the (target_ name, key_name, context_handle) mapping table, return to the uninitialized state and SHOULD negotiate a new security context, as described above in Section 3.1

If major_status is GSS_S_BAD_QOP, the caller SHOULD repeat the GSS_GetMIC call with allowed QOP value. The number of such repetitions MUST be limited to prevent infinite loops.

5.2. Verifying a Signed Message - Call GSS_VerifyMIC

The procedure for verifying a signature-protected message is specified in [RFC2845].

The NAME of the TSIG record determines which context_handle maps to the context that MUST be used to verify the signature. If the NAME does not map to an established context, the server MUST send a standard TSIG error response to the client indicating BADKEY in the TSIG error field (as described in [RFC2845]).

For the GSS algorithm, a signature is verified by using GSS_VerifyMIC:

```
INPUTS
  CONTEXT HANDLE context_handle = context_handle for key_name
  OCTET STRING   message        = incoming message plus TSIG
                                  variables (per [RFC2845])
  OCTET STRING   per_msg_token  = Signature field from TSIG RR

OUTPUTS
  INTEGER        major_status
  INTEGER        minor_status
  INTEGER        qop_state
```

If major_status is GSS_S_COMPLETE, the signature is authentic and the message was delivered intact. Per [RFC2845], the timer values of the TSIG record MUST also be valid before considering the message to be authentic. The caller MUST not act on the request or response in the message until these checks are verified.

When a server is processing a client request, the server MUST send a standard TSIG error response to the client indicating BADKEY in the TSIG error field as described in [RFC2845], if major_status is set to one of the following values

```
GSS_S_DEFECTIVE_TOKEN
GSS_S_BAD_SIG (GSS_S_BAD_MIC)
GSS_S_DUPLICATE_TOKEN
GSS_S_OLD_TOKEN
GSS_S_UNSEQ_TOKEN
GSS_S_GAP_TOKEN
GSS_S_CONTEXT_EXPIRED
GSS_S_NO_CONTEXT
GSS_S_FAILURE
```

If the timer values of the TSIG record are invalid, the message MUST NOT be considered authentic. If this error checking fails when a server is processing a client request, the appropriate error response MUST be sent to the client according to [RFC2845].

6. Example usage of GSS-TSIG algorithm

This Section describes an example where a Client, client.example.com, and a Server, server.example.com, establish a security context according to the algorithm described above.

I. Client initializes security context negotiation

To establish a security context with a server, server.example.com, the Client calls GSS_Init_sec_context with the following parameters. (Note that some INPUT and OUTPUT parameters not critical for this algorithm are not described in this example.)

```
CONTEXT HANDLE input_context_handle  = 0
INTERNAL NAME  targ_name             = "DNS@server.example.com"
OCTET STRING   input_token           = NULL
BOOLEAN        replay_det_req_flag   = TRUE
BOOLEAN        mutual_req_flag       = TRUE
```

The OUTPUTS parameters returned by GSS_Init_sec_context include

```
INTEGER        major_status = GSS_S_CONTINUE_NEEDED
CONTEXT HANDLE output_context_handle context_handle
OCTET STRING   output_token output_token
BOOLEAN        replay_det_state = TRUE
BOOLEAN        mutual_state = TRUE
```

Client verifies that replay_det_state and mutual_state values are TRUE. Since the major_status is GSS_S_CONTINUE_NEEDED, which is a success OUTPUT major_status value, client stores context_handle that maps to "DNS@server.example.com" and proceeds to the next step.

II. Client sends a query with QTYPE = TKEY to server

Client sends a query with QTYPE = TKEY for a client-generated globally unique domain name string, 789.client.example.com.server.example.com. Query contains a TKEY record in its Additional records section with the following fields. (Note that some fields not specific to this algorithm are not specified.)

```
NAME = 789.client.example.com.server.example.com.
RDATA
   Algorithm Name      = gss-tsig
   Mode                = 3 (GSS-API negotiation - per [RFC2930])
   Key Size            = size of output_token in octets
   Key Data            = output_token
```

After the key_name 789.client.example.com.server.example.com. is generated it is stored in the client's (target_name, key_name, context_handle) mapping table.

III. Server receives a query with QTYPE = TKEY

When server receives a query with QTYPE = TKEY, the server verifies that Mode and Algorithm fields in the TKEY record in the Additional records section of the query are set to 3 and "gss-tsig" respectively. It finds that the key_name 789.client.example.com.server.example.com. is not listed in its (key_name, context_handle) mapping table.

IV. Server calls GSS_Accept_sec_context

To continue security context negotiation server calls GSS_Accept_sec_context with the following parameters. (Note that some INPUT and OUTPUT parameters not critical for this algorithm are not described in this example.)

```
 INPUTS
   CONTEXT HANDLE input_context_handle  = 0
   OCTET STRING   input_token           = token specified in the Key
                            field from TKEY RR (from Additional
                            records section of the client's query)
```

The OUTPUTS parameters returned by GSS_Accept_sec_context include

```
   INTEGER        major_status = GSS_S_CONTINUE_NEEDED
   CONTEXT_HANDLE output_context_handle context_handle
   OCTET STRING   output_token output_token
```

Server stores the mapping of the 789.client.example.com.server.example.com. to OUTPUT context_handle in its (key_name, context_handle) mapping table.

V. Server responds to the TKEY query

Since the major_status = GSS_S_CONTINUE_NEEDED in the last server's call to GSS_Accept_sec_context, the server responds to the TKEY query placing in the answer section a TKEY record containing output_token in the Key Data RDATA field. The error field in the TKEY record is set to 0. The RCODE in the query response is set to NOERROR.

VI. Client processes token returned by server

When the client receives the TKEY query response from the server, the client calls GSS_Init_sec_context with the following parameters. (Note that some INPUT and OUTPUT parameters not critical for this algorithm are not described in this example.)

```
   CONTEXT HANDLE input_context_handle  = the context_handle stored
        in the client's mapping table entry (DNS@server.example.com.,
        789.client.example.com.server.example.com., context_handle)
   INTERNAL NAME  targ_name             = "DNS@server.example.com"
   OCTET STRING   input_token           = token from Key field of TKEY
        record from the Answer section of the server's response
   BOOLEAN        replay_det_req_flag   = TRUE
   BOOLEAN        mutual_req_flag       = TRUE
```

The OUTPUTS parameters returned by GSS_Init_sec_context include

```
   INTEGER        major_status = GSS_S_COMPLETE
   CONTEXT HANDLE output_context_handle = context_handle
   OCTET STRING   output_token = output_token
   BOOLEAN        replay_det_state = TRUE
   BOOLEAN        mutual_state = TRUE
```

Since the major_status is set to GSS_S_COMPLETE the client side security context is established, but since the output_token is not NULL client MUST send a TKEY query to the server as described below.

VII. Client sends a query with QTYPE = TKEY to server

Client sends to the server a TKEY query for the 789.client.example.com.server.example.com. name. Query contains a TKEY record in its Additional records section with the following fields. (Note that some INPUT and OUTPUT parameters not critical to this algorithm are not described in this example.)

```
   NAME = 789.client.example.com.server.example.com.
   RDATA
      Algorithm Name      = gss-tsig
      Mode                = 3 (GSS-API negotiation - per [RFC2930])
      Key Size            = size of output_token in octets
      Key Data            = output_token
```

VIII. Server receives a TKEY query

When the server receives a TKEY query, the server verifies that Mode and Algorithm fields in the TKEY record in the Additional records section of the query are set to 3 and gss-tsig, respectively. It finds that the key_name 789.client.example.com.server.example.com. is listed in its (key_name, context_handle) mapping table.

IX. Server calls GSS_Accept_sec_context

To continue security context negotiation server calls GSS_Accept_sec_context with the following parameters (Note that some INPUT and OUTPUT parameters not critical for this algorithm are not described in this example)

```
 INPUTS
   CONTEXT HANDLE input_context_handle  = context_handle from the
         (789.client.example.com.server.example.com., context_handle)
         entry in the server's mapping table
   OCTET STRING   input_token           = token specified in the Key
         field of TKEY RR (from Additional records Section of
         the client's query)
```

The OUTPUTS parameters returned by GSS_Accept_sec_context include

```
   INTEGER        major_status = GSS_S_COMPLETE
   CONTEXT_HANDLE output_context_handle = context_handle
   OCTET STRING   output_token = NULL
```

Since major_status = GSS_S_COMPLETE, the security context on the server side is established, but the server still needs to respond to the client's TKEY query, as described below. The security context state is advanced to Context Established.

X. Server responds to the TKEY query

Since the major_status = GSS_S_COMPLETE in the last server's call to GSS_Accept_sec_context and the output_token is NULL, the server responds to the TKEY query placing in the answer section a TKEY record that was sent by the client in the Additional records section of the client's latest TKEY query. In addition, this server places a TSIG record in additional records section of its response. Server calls GSS_GetMIC to generate a signature to include it in the TSIG record. The server specifies the following GSS_GetMIC INPUT parameters:

```
   CONTEXT HANDLE context_handle = context_handle from the
         (789.client.example.com.server.example.com., context_handle)
         entry in the server's mapping table
   OCTET STRING   message        = outgoing message plus TSIG
                                   variables (as described in [RFC2845])
```

The OUTPUTS parameters returned by GSS_GetMIC include

```
   INTEGER        major_status = GSS_S_COMPLETE
   OCTET STRING   per_msg_token
```

Signature field in the TSIG record is set to per_msg_token.

XI. Client processes token returned by server

Client receives the TKEY query response from the server. Since the major_status was GSS_S_COMPLETE in the last client's call to GSS_Init_sec_context, the client verifies that the server's response is signed. To validate the signature, the client calls GSS_VerifyMIC with the following parameters:

```
 INPUTS
   CONTEXT HANDLE context_handle = context_handle for
                789.client.example.com.server.example.com. key_name
   OCTET STRING   message        = incoming message plus TSIG
                               variables (as described in [RFC2845])
   OCTET STRING   per_msg_token  = Signature field from TSIG RR
                included in the server's query response
```

Since the OUTPUTS parameter major_status = GSS_S_COMPLETE, the signature is validated, security negotiation is complete and the security context state is advanced to Context Established. These client and server will use the established security context to sign and validate the signatures when they exchange packets with each other until the context expires.

7. Security Considerations

This document describes a protocol for DNS security using GSS-API. The security provided by this protocol is only as effective as the security provided by the underlying GSS mechanisms.

All the security considerations from RFC 2845, RFC 2930 and RFC 2743 apply to the protocol described in this document.

8. IANA Considerations

The IANA has reserved the TSIG Algorithm name gss-tsig for the use in the Algorithm fields of TKEY and TSIG resource records. This Algorithm name refers to the algorithm described in this document. The requirement to have this name registered with IANA is specified in RFC 2845.

9. Conformance

The GSS API using SPNEGO [RFC2478] provides maximum flexibility to choose the underlying security mechanisms that enables security context negotiation. GSS API using SPNEGO [RFC2478] enables client and server to negotiate and choose such underlying security mechanisms on the fly. To support such flexibility, DNS clients and servers SHOULD specify SPNEGO mech_type in their GSS API calls. At the same time, in order to guarantee interoperability between DNS clients and servers that support GSS-TSIG it is required that

- DNS servers specify SPNEGO mech_type
- GSS APIs called by DNS client support Kerberos v5
- GSS APIs called by DNS server support SPNEGO [RFC2478] and Kerberos v5.

In addition to these, GSS APIs used by DNS client and server MAY also support other underlying security mechanisms.

10. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

11. Acknowledgements

The authors of this document would like to thank the following people for their contribution to this specification: Chuck Chan, Mike Swift, Ram Viswanathan, Olafur Gudmundsson, Donald E. Eastlake, 3rd and Erik Nordmark.

12. References

12.1. Normative References

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

[RFC2478] Baize, E. and D. Pinkas, "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface, Version 2 , Update 1", RFC 2743, January 2000.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D. and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2930] Eastlake 3rd, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

12.2. Informative References

[ISO11578] "Information technology", "Open Systems Interconnection", "Remote Procedure Call", ISO/IEC 11578:1996, http://www.iso.ch/cate/d2229.html.

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1034, November 1987.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[RFC2137] Eastlake 3rd, D., "Secure Domain Name System Dynamic Update", RFC 2137, April 1997.

[RFC2535] Eastlake 3rd, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

13. Authors' Addresses

Stuart Kwan
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA
EMail: skwan@microsoft.com

Praerit Garg
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA
EMail: praeritg@microsoft.com

James Gilroy
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA
EMail: jamesg@microsoft.com

Levon Esibov
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA
EMail: levone@microsoft.com

Randy Hall
Lucent Technologies
400 Lapp Road
Malvern PA 19355
USA
EMail: randyhall@lucent.com

Jeff Westhead
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA
EMail: jwesth@microsoft.com

14. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.