Stuart Kwan
Praerit Garg
James Gilroy
Microsoft Corp.

February 2000
<draft-skwan-gss-tsig-05.txt>
Expires August 2000

GSS Algorithm for TSIG (GSS-TSIG)

Status of this Memo

Abstract

The 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) [RFC2078].

Table of Contents

1: Introduction...2
2: Protocol Overview..3
  2.1: GSS Details..4
3: Client Protocol Details..4
  3.1: Negotiating Context..4
    3.1.1: Call GSS_Init_sec_context................................5
    3.1.2: Send TKEY Query to Server................................6
    3.1.3: Receive TKEY Query-Response from Server..................6
  3.2: Context Established..8
4: Server Protocol Details..8
  4.1: Negotiating Context..8
    4.1.1: Receive TKEY Query from Client...........................9
    4.1.2: Call GSS_Accept_sec_context..............................9
    4.1.3: Send TKEY Query-Response to Client.......................9
  4.2: Context Established...10
    4.2.1: Terminating a Context...................................10
5: Sending and Verifying Signed Messages...........................11
  5.1: Sending a Signed Message - Call GSS_GetMIC..................11
  5.2: Verifying a Signed Message - Call GSS_VerifyMIC.............12
6: Security Considerations...12
7: Acknowledgements..13
8: References..13

1. Introduction

The Secret Key Transaction Signature for DNS [TSIG] protocol was developed to provide a lightweight alternative to full DNS security [RFC2535] and secure dynamic update [RFC2137], where full security is impractical due to implementation complexity, management overhead, or computational cost.

The [TSIG] protocol is extensible through the definition of new algorithms. This document specifies an algorithm based on the Generic Security Service Application Program Interface (GSS-API) [RFC2078]. 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 for one transaction, whereas that same server may use TLS 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.

2. Protocol Overview

Readers that are unfamiliar with the GSS-API concepts are encouraged to read the characteristics and concepts section of [RFC2078] before examining this protocol in detail.

In GSS, client and server interact to create a "security context". The security context is 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 per [RFC2078] before using this protocol.

Not all flags used in GSS-API interfaces are specified in this document. Where omitted, clients and servers may select the default or use a value based on local system policy.

Not all error return values from GSS-API interfaces are specified in this document. When errors are encountered, the caller should act appropriately.

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 handles to servers,

```
     (target_server_name, key_name, context_handle)
```

The value key_name also identifies a context handle, and is used on the wire to indicate to a server which context should be used to process the current request.

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 [TKEY] 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 [TKEY].

3.1.1 Call GSS_Init_sec_context

The client obtains the first token by calling GSS_Init_sec_context. The following input parameters are used. The outcome of the call is indicated with the output values below. Consult [RFC2078] for syntax definitions.

```
   INPUTS
     CONTEXT HANDLE input_context_handle  = 0
     INTERNAL NAME  targ_name             = DNS/<target_server_name>
     OCTET STRING   input_token           = NULL
     BOOLEAN        replay_det_req_flag   = TRUE
     BOOLEAN        mutual_req_flag       = TRUE
     BOOLEAN        deleg_req_flag        = TRUE (optional)

   OUTPUTS
     INTEGER        major_status
     CONTEXT HANDLE output_context_handle
     OCTET STRING   output_token
     BOOLEAN        replay_det_state
     BOOLEAN        mutual_state
```

The values of replay_det_state and mutual_state indicate if the security package can provide replay detection and mutual authentication, respectively. If one or both of these values are FALSE, the client must abandon this protocol.

The deleg_req_flag is optional, and can be used if the client wants the server to be able to call out to other services under the context of the client.

If major_status indicates an error, the client must abandon the protocol. Success values of major_status are GSS_S_CONTINUE_NEEDED and GSS_S_COMPLETE. The exact success code is important during later processing.

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_server_name.

3.1.2 Send TKEY Query to Server

The output_token from GSS_Init_sec_context is transmitted to the server in a query request for a TKEY record. The token itself will be placed in a TKEY record in the additional records section of the query. The domain-like name of the TKEY record set queried for and the name of the supplied TKEY record in the additional section will uniquely identify the security context to both the client and server, and thus the client should use a value which is globally unique.

```
   TKEY Record
     NAME = client-generated globally unique domain name string
            (see [TKEY])
     RDATA
        Algorithm Name      = gss-tsig.microsoft.com
        Mode                = 3 (GSS-API negotiation - see [TKEY])
        Key Size            = size of output_token
        Key                 = output_token
```

Assign the remaining fields in the TKEY RDATA appropriate values per [TKEY].

If the last call to GSS_Init_sec_context yielded a major_status value of GSS_S_COMPLETE, then the message should be signed with a TSIG record before being sent to the server. See section 5, Sending and Verifying Signed Messages, for the signing procedure.

The query is transmitted to the server.

3.1.3 Receive TKEY Query-Response from Server

The server will return a standard TKEY query-response (see [TKEY]). The response may indicate that TKEY is not supported, or that the GSS-API mode and algorithm are not supported. If this is the case, the client must abandon this algorithm.

If the value of the Error field in the TKEY RDATA (TKEY.Error) is BADKEY, then the token provided by the client was invalid. The client must abandon this algorithm.

If TKEY.Error indicates success, then the client may continue. The next processing step depends on the value of major_status from the most recent call 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 will be in the form of a NOERROR query response containing the last client supplied TKEY record in the answer section of the query. The response may also be signed with a TSIG record, and if present this signature must be verified using the procedure detailed in section 5, Sending and Verifying Signed Messages.

If the message verification completes without an error, or if a TSIG signature was not included, 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

If the last call to GSS_Init_sec_context yielded a major_status value of GSS_S_CONTINUE, then the negotiation is not yet complete. The server will respond to the TKEY query with a NOERROR query response that contains a TKEY record in the answer section. The TKEY record contains a token that is passed to GSS_Init_sec_context using the parameters from the previous call and the following modifications:

```
   INPUTS
     CONTEXT HANDLE input_context_handle  = context_handle
     OCTET STRING   input_token           = token from Key field of
                                            TKEY record

   OUTPUTS
     INTEGER        major_status
     OCTET STRING   output_token
```

If major_status indicates an error, the client must abandon this algorithm. Success values are GSS_S_CONTINUE and GSS_S_COMPLETE.

If major_status is GSS_S_CONTINUE the negotiation is not yet finished. The token output_token must again be passed to the server in a TKEY record. The negotiation sequence is repeated beginning with section 3.1.2. The client should place a limit on the number of continuations in a context negotiation to prevent endless looping.

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. The negotiation sequence is repeated beginning with section 3.1.2.

If major_status is GSS_S_COMPLETE and output_token is NULL, context negotiation is complete. The response from the server may be signed with a TSIG record, and if present this signature must be verified using the procedure detailed in section 5, Sending and Verifying Signed Messages.

If the message verification completes without an error, or if a TSIG signature was not included, 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 is used for the generation and verification of transaction signatures.

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

4. Server Protocol Details

As on the client-side, the result of a successful context negotiation is a context handle used in future processing.

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 recognizes TKEY queries as security context negotiation messages.

4.1.1 Receive TKEY Query from Client

Upon receiving a TKEY query, the server must examine the Mode and Algorithm Name fields to see if they match this algorithm. If they match, the (key_name, context_handle) mapping table is searched for the NAME value of the TKEY record. If the name is found in the table, the corresponding context_handle is used in following GSS operations. If the name is not found a new negotiation is started.

4.1.2 Call GSS_Accept_sec_context

The server performs its side of a context negotiation by calling GSS_Accept_sec_context with the token provided by the client in the TKEY record.

The server calls GSS_Accept_sec_context:

```
   INPUTS
     CONTEXT HANDLE input_context_handle  = 0 if new negotiation,
                                            context_handle if ongoing
     OCTET STRING   input_token           = Key field from TKEY RR

   OUTPUTS
     INTEGER        major_status
     CONTEXT_HANDLE output_context_handle
     OCTET STRING   output_token
```

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

If major_status returns a GSS failure code, the negotiation has failed. The server must respond to the client with a standard TKEY query-response where the TKEY error field value is set to BADKEY.

Success values for major_status are GSS_S_COMPLETE or GSS_S_CONTINUE.

If major_status is GSS_S_COMPLETE the server component of the negotiation is finished. The message from the client may be signed with a TSIG RR, and if present this signature must be verified using the procedure detailed in section 5, Sending and Verifying Signed Messages. The server responds to the TKEY query using a standard query response. If output_token is non-NULL, then it must be returned to the client in a TKEY in the Answer section of the response. If output_token is NULL, then the TKEY record received from the client must be returned in the Answer section of the response. The answer should be signed with a TSIG record per the procedure given in section 5, Sending and Verifying Signed Messages. The context state is advanced to Established. Section 4.2 discusses the usage of the security context.

If major_status is GSS_S_CONTINUE, the server component of the negotiation is not yet finished. The server responds to the TKEY query with a standard query response, placing a TKEY record containing output_token in the answer section. The negotiation sequence then repeats beginning with section 4.1.1. The server must limit the number of times that a given context is allowed to repeat, to prevent endless looping.

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.

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 "key deletion". See [TKEY] for more details.

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 [TSIG]. The data to be passed to the signature routine includes the whole DNS message with specific TSIG variables appended. For the exact format, see [TSIG]. For this protocol, use the following TSIG variable values:

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

Assign the remaining fields in the TKEY RDATA appropriate values per [TKEY].

For the GSS algorithm, the signature is generated by calling GSS_GetMIC:

```
   INPUTS
     CONTEXT HANDLE context_handle = context_handle for key_name
     OCTET STRING   message        = outgoing message plus TSIG
                                     variables (see [TSIG])

   OUTPUTS
     INTEGER        major_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 or GSS_S_CREDENTIALS_EXPIRED, the caller needs to return to the uninitialized state and negotiate a new security context.

5.2 Verifying a Signed Message - Call GSS_VerifyMIC

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

The NAME of the TSIG record determines which context_handle maps to this context. If the NAME does not map to a currently active context, the server must send a standard TSIG error response to the client indicating BADKEY in the TSIG error field (see [TSIG]).

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 (see [TSIG])
     OCTET STRING   per_msg_token  = Signature field from TSIG RR

   OUTPUTS
     INTEGER        major_status
```

If major_status is GSS_S_COMPLETE, the signature is authentic and the message was delivered intact. Per [TSIG], 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.

If major_status is GSS_S_CONTEXT_EXPIRED, the negotiated context is no longer valid. If this failure occurs 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 (see [TSIG]).

If major_status is any other error code or 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 per [TSIG].

6. 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.

7. 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 and Donald E. Eastlake 3rd.

8. References

[RFC2078] J. Linn, "Generic Security Service Application Program Interface, Version 2," RFC 2078, OpenVision Technologies, January 1997.

[TSIG] P. Vixie, O. Gudmundsson, D. Eastlake, "Secret Key Transaction Signatures for DNS (TSIG)," draft-ietf-dnsind-tsig-*..txt, ISC, TIS, Cybercash.

[TKEY] D. Eastlake 3rd, "Secret Key Establishment for DNS (TKEY RR)," draft-ietf-dnssec-tkey-*.txt.

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

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

9. Author's Addresses

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

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

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