

PPPEXT Working Group
INTERNET-DRAFT
Category: Experimental
<[draft-aboba-pppext-eapgss-12.txt](#)>
[6](#) April 2002

Bernard Aboba
Dan Simon
Microsoft

EAP GSS Authentication Protocol

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC 2026](#).

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

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

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

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

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

The Extensible Authentication Protocol (EAP) provides a standard mechanism for support of multiple authentication methods, including public key, smart cards, One Time Passwords, and others.

This document describes the EAP GSS protocol, which enables the use of GSS-API mechanisms within EAP. As a result, any GSS-API mechanism providing initial authentication can be used with EAP GSS.

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

Table of Contents

1.	Introduction	3
1.1	Requirements language	3
1.2	Terminology	3
2.	Protocol overview	4
2.1	EAP server as GSS-API initiator	4
2.2	Peer as GSS-API initiator	6
3.	Detailed description of EAP GSS protocol	9
3.1	EAP GSS packet format	9
3.2	EAP GSS Request packet	10
3.3	EAP GSS Response packet	12
3.4	Options	13
3.5	Fragmentation	15
3.6	Retry behavior	18
3.7	Identity verification	18
3.8	Use of addresses	19
4.	Key management	19
4.1	Key derivation algorithm	19
5.	Security considerations	21
5.1	Dictionary attacks	21
5.2	Certificate revocation	23
5.3	Mutual authentication	23
5.4	Credential reuse	23
5.5	Key transmission issues	25
5.6	Protected negotiation	26
6.	Normative References	27
7.	Informative References	28
	IANA Considerations	30
	Acknowledgments	30
	Author's Addresses	30
	Appendix A - Example IAKERB topologies	31
A.1	RADIUS+KDC backend	31
A.2	Kerberos KDC backend	34
	Appendix B - The Pseudorandom Function	36
	Intellectual Property Statement	38
	Full Copyright Statement	38

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

[1.](#) Introduction

The Extensible Authentication Protocol (EAP) [[RFC2284](#)] provides a standard mechanism for support of multiple authentication methods, including public key [[RFC2716](#)], smart cards, One Time Passwords [[RFC2284](#)], and others. EAP may run directly over the link layer without requiring IP and therefore includes its own support for in-order delivery and re-transmission, but not fragmentation and reassembly, which is the responsibility of individual EAP methods. While EAP was originally developed for use with PPP [[RFC1661](#)], it is also now in use with IEEE 802 [[IEEE802](#)]. The encapsulation of EAP on IEEE 802 links is described in [[IEEE8021X](#)].

The Generic Security Service API (GSS-API), is described in [[RFC2743](#)]. This document describes the EAP GSS protocol, which supports fragmentation and reassembly and enables the use of GSS-API mechanisms within EAP. As a result, any GSS-API mechanism providing initial authentication can be used with EAP GSS.

Supporting GSS-API authentication methods within EAP is desirable because this enables developers creating GSS-API authentication methods to leverage their development efforts. Since the EAP Type field is a finite (one octet) resource, EAP GSS allows GSS-API methods to automatically be supported within EAP without having to consume an EAP Type for each GSS-API method.

[1.1.](#) Requirements language

In this document, the key words "MAY", "MUST", "MUST NOT", "OPTIONAL", "RECOMMENDED", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [[RFC2119](#)].

[1.2.](#) Terminology

This document frequently uses the following terms:

Authentication Server

An Authentication Server is an entity that provides an Authentication Service to an NAS. This service verifies from the credentials provided by the peer, the claim of identity made by the peer.

Master key

The key derived between the EAP client and EAP server during the EAP authentication process.

Master session key

The keys derived from the master key that are subsequently

Aboba

Experimental

[Page 3]

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

used in generation of the transient session keys for authentication, encryption, and IV-generation. So that the master session keys are usable with any ciphersuite, they are longer than is necessary, and are truncated to fit.

NAS

The end of the link requiring the authentication. In IEEE 802.1X, this end is known as the Authenticator.

Peer

The other end of the point-to-point link (PPP), point-to-point LAN segment (IEEE 802.1X) or 802.11 wireless link, which being authenticated by the NAS. In IEEE 802.1X, this end is known as the Supplicant.

Transient session keys

The chosen ciphersuites uses transient session keys for authentication and encryption as well as IVs (if required). The transient session keys are derived from the master session keys, and are of the appropriate size and type for use with the chosen ciphersuite.

[2.](#) Protocol overview

As described in [[RFC2284](#)], EAP conversations will typically begin with the NAS and the peer negotiating EAP. The NAS will then typically send an EAP-Request/Identity packet to the peer, and the peer will respond with an EAP-Response/Identity packet to the NAS, containing the peer's UserId. Once having received the peer's Identity, the EAP server responds with an EAP-Request packet of EAP-Type=EAP GSS. From this point forward, the EAP GSS conversation may proceed in one of two ways:

- [1] EAP server as GSS-API initiator. Here the EAP server acts as the GSS-API initiator, and the peer acts as the GSS-API target.
- [2] EAP client as GSS-API initiator. Here the peer acts as the GSS-API initiator, and the EAP server acts as the GSS-API target. This mode adds an extra round-trip.

2.1. EAP server as GSS-API initiator

As described in [[RFC2284](#)], the EAP server typically authenticates the peer using a prearranged method or set of methods. As a result, the EAP server may have predetermined the use of EAP GSS as well as the GSS-API method to be used. If that GSS-API method can be initiated by the EAP Server, then the EAP server MAY act as a GSS-API initiator with the peer acting as a GSS-API target. In this case, the EAP Server will indicate the pre-determined GSS-API method, possibly via SPNEGO, but SHOULD NOT allow negotiation of a substitute GSS-API method.

To initiate the conversation, the EAP-Server sends an EAP-Request packet with EAP-Type=EAP GSS. This initial packet MUST have the 0 (Options) bit set, and MUST include a Nonce Payload option. The data field of the packet will encapsulate a GSS-API token, created as a result of a call to `GSS_Init_sec_context()`. In this case mutual authentication MUST be requested (otherwise the peer would not be authenticated to the NAS!) so that the `mutual_req_flag` is set and the call to `GSS_Init_sec_context()` returns `GSS_S_CONTINUE_NEEDED` status.

When it receives the EAP-Request, the peer will de-capsulate the received GSS-API token within the EAP GSS frame, and will pass it as the `input_token` parameter to `GSS_Accept_sec_context()`. If `GSS_Accept_sec_context` indicates `GSS_S_COMPLETE` status, then the NAS has been authenticated by the peer, and the NAS's indicated identity is provided in the `src_name` result. The peer then responds with an EAP-Response packet with EAP-Type= EAP GSS, MUST have the 0 bit (Options) set, and contains a Nonce Payload option. The data field of the packet contains the returned `output_token`.

The EAP server will then de-capsulate the GSS-API token within the EAP-Response message and pass it as the `input_token` parameter to `GSS_Init_sec_context()`. If the call returns `GSS_S_COMPLETE` status, then

the peer has been authenticated to the EAP-Server, then the EAP-Server responds with an EAP-Success message. If GSS_S_CONTINUE_NEEDED status is returned, then the EAP Server encapsulates the returned output_token with an EAP-Request packet of EAP-Type=EAP GSS, and pass this back to the peer.

The conversation (which can be completed in a minimum of 2.5 round trips), appears as follows:

```
Peer          NAS
-----
                EAP/Identity
                <-----Request
```

```
EAP/Identity
Response ----->
```

```
GSS_Init_sec_context(mutual_req_flag)
returns GSS_S_CONTINUE_NEEDED,
output_token
```

```
<-----EAP Request
      EAP Type=EAP GSS
      0 bit, Nonce Payload,
      output_token
```

```
GSS_Accept_sec_context(input_token)
returns GSS_S_COMPLETE,
output_token
```

```
EAP Response ----->
EAP Type=EAP GSS
0 bit, Nonce Payload,
output_token
```

```
GSS_Init_sec_context(input_token)
returns GSS_S_COMPLETE
```

```
<-----EAP Success
```

[2.2.](#) Peer as GSS-API initiator

If the EAP server is prepared to allow negotiation of the GSS-API method via SPNEGO [[RFC2478](#)], or if the EAP server knows the GSS-API method to be used, but cannot initiate it (e.g. IAKERB, or Kerberos V), then the peer MUST act as a GSS-API initiator, with the EAP server acting as the GSS-API target.

In this case, the EAP server MUST respond with an EAP GSS/Start packet, which is an EAP-Request packet with EAP-Type=EAP GSS, the Start (S) and Options (O) bits set, a Nonce Payload option, and no data.

The peer then calls GSS_Init_sec_context(), typically with mutual authentication requested so that the mutual_req_flag is set and the call

returns GSS_S_CONTINUE_NEEDED status. The peer then MUST respond with an EAP-Response packet with EAP-Type=EAP GSS, the 0 bit set, the Nonce Payload option present, and the data field set to the output_token.

If method negotiation is to be used, then an initial negotiation token for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) [[RFC2478](#)] is transferred. This contains an ordered list of mechanisms, a set of options that should be supported by the selected mechanism and

the initial security token for the mechanism preferred by the peer. The inclusion of the initial security token for the preferred method saves a round-trip, assuming that the NAS agrees to the preferred mechanism.

The EAP server then de-capsulates the GSS-API token contained within the EAP-Response of EAP-Type=EAP GSS and uses this as the `input_token` parameter to a call to `GSS_Accept_sec_context()`. The `output_token` parameter will then contain a token, containing the result of the negotiation and in the case of accept, the agreed security mechanism and the response to the initial security token as described in [[RFC2478](#)]. This token is then encapsulated within an EAP-Request packet of EAP-Type=GSS-API, which is sent to the peer. This occurs whether the call completed with `GSS_S_CONTINUE_NEEDED` status or `GSS_S_COMPLETE` status.

The peer then de-capsulates the GSS-API token contained within the EAP-Request packet with EAP-Type=EAP GSS, and passes the `input_token` parameter to `GSS_Init_sec_context()`. The `output_token` is encapsulated within an EAP-Response packet with EAP-Type=EAP GSS and sent to the EAP server. This occurs whether the call completed with `GSS_S_CONTINUE_NEEDED` status or `GSS_S_COMPLETE` status.

If the previous call to `GSS_Accept_sec_context()` returned `GSS_S_COMPLETE` status, then the EAP-Server returns an EAP-Success message to the client. Otherwise, it de-capsulates the GSS-API token contained within the EAP-Request packet, and the conversation continues.

trips), appears as follows:

```
Authenticating Peer      NAS
-----
EAP-Request/
<- Identity

EAP-Response/
Identity (MyID) ->

EAP-Request/
EAP-Type=EAP GSS
(GSS Start,
  S and 0 bits set),
<- Nonce Payload

GSS_Init_sec_context(mutual_req_flag)
  returns GSS_S_CONTINUE_NEEDED,
  output_token (SPNEGO)

EAP-Response/
EAP-Type=EAP GSS
0 bit, Nonce Payload,
output_token ->

GSS_Accept_sec_context(input_token)
  returns GSS_S_COMPLETE,
  output_token (SPNEGO)

EAP-Request/
EAP-Type=EAP GSS
<- output_token

GSS_Init_sec_context(input_token)
  returns GSS_S_COMPLETE,
  output_token

EAP-Response/
EAP-Type=EAP GSS
output_token ->

<- EAP-Success
```

3.1. EAP GSS Packet Format

[illegible]

- 1 - Request
- 2 - Response

The identifier field is one octet and aids in matching responses with requests.

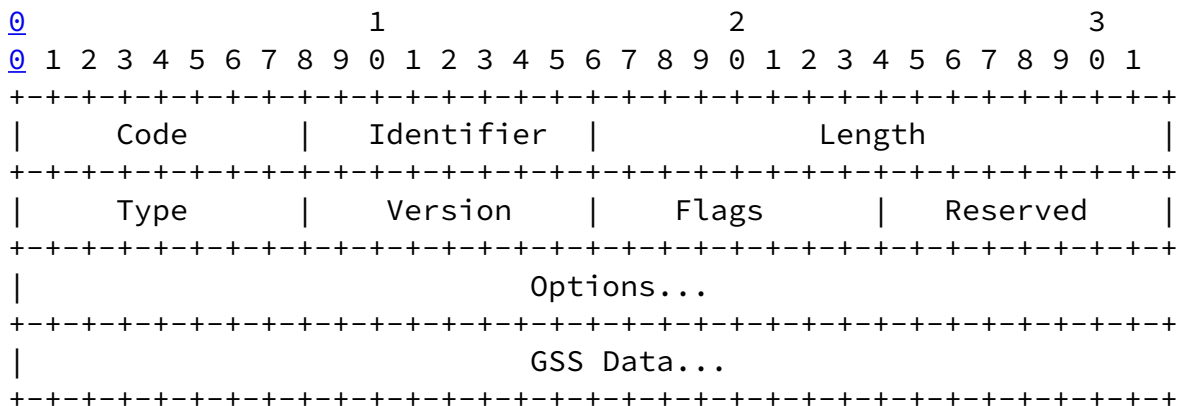
The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, and Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and should be ignored on reception.

TBD - EAP GSS

The format of the Data field is determined by the Code field.

[3.2.](#) EAP GSS Request Packet

A summary of the EAP GSS Request packet format is shown below. The fields are transmitted from left to right.



Code

1

Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field **MUST** be changed on each Request packet.

Length

The Length field is two octets and indicates the length of the entire EAP packet.

Type

TBD - EAP GSS

Version

1

Flags

```
0 1 2 3 4 5 6 7 8
+---+---+---+---+
| L M S O R R R R |
+---+---+---+---+
```

L = Length included
M = More fragments
S = EAP GSS start
O = Options present
R = Reserved

The L bit (length included) MUST be set for the first fragment of a fragmented GSS message or set of messages. If set, the GSS Message Length option MUST be included. The M bit (more fragments) is set on all but the last fragment. The S bit (EAP GSS start) is set in an EAP GSS Start message. This differentiates the EAP GSS Start message from a fragment acknowledgment. The O bit (Options present) is set to indicate the presence of options, including the GSS Message Length option. As a result, the O bit MUST be set whenever the L bit is set; however, it also may be set when the L bit is not set.

Options

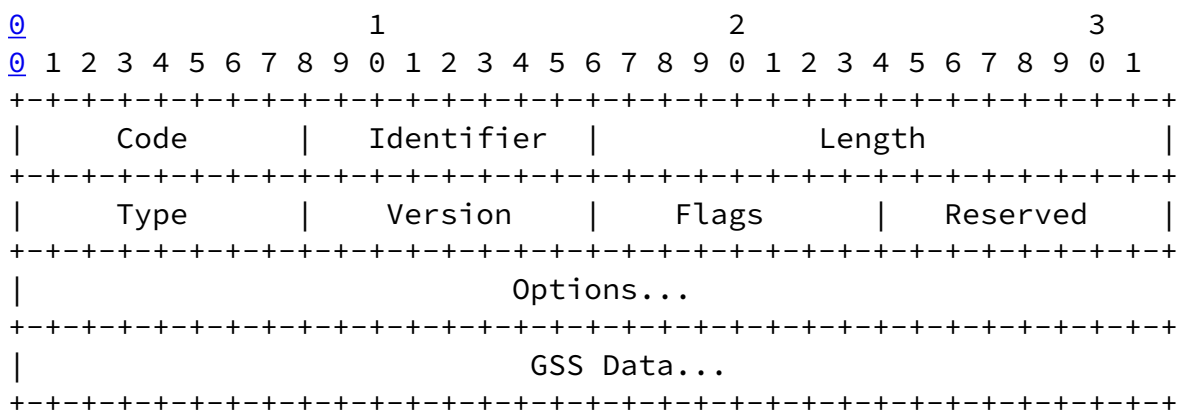
The Options field is of variable length, and contains type-length-value tuples (TLVs). Options are present only if the O bit is set.

GSS data

The GSS data consists of the encapsulated GSS token.

[3.3.](#) EAP GSS Response Packet

A summary of the EAP GSS Response packet format is shown below. The fields are transmitted from left to right.



Code

2

Identifier

The Identifier field is one octet and MUST match the Identifier field from the corresponding request.

Length

The Length field is two octets and indicates the length of the entire EAP packet.

Type

TBD - EAP GSS

Version

1

Aboba

Experimental

[Page 12]

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

Flags

```
0 1 2 3 4 5 6 7 8
+---+---+---+---+
| L M S O R R R R |
+---+---+---+---+
```

L = Length included
M = More fragments
S = EAP GSS start
O = Options present
R = Reserved

The L bit (length included) MUST be set for the first fragment of a fragmented GSS message or set of messages. If set, the GSS Message Length option MUST be included. The M bit (more fragments) is set on all but the last fragment. The S bit (EAP GSS start) is set in an EAP GSS Start message. This differentiates the EAP GSS Start message

from a fragment acknowledgment. The 0 bit (Options present) is set to indicate the presence of options, including the GSS Message Length option. As a result, the 0 bit MUST be set whenever the L bit is set; however, it also may be set when the L bit is not set.

Options

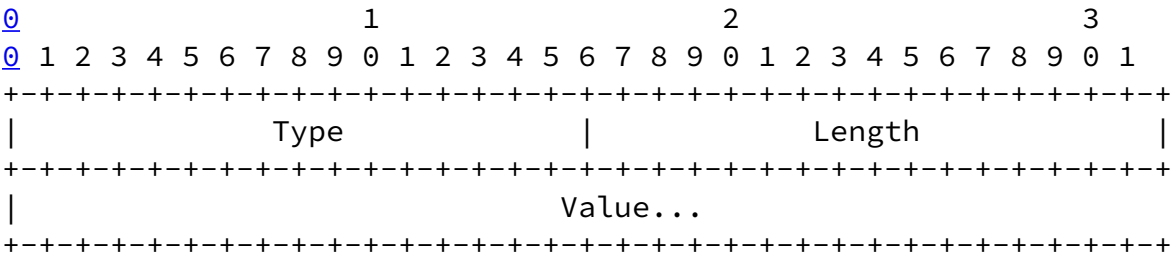
The Options field is of variable length, and contains type-length-value tuples (TLVs). Options are present only if the 0 bit is set.

GSS data

The GSS data consists of the encapsulated GSS token.

3.4. Options

To date, the only options that are defined are the GSS Message Length option (option 1) and the Nonce Payload option (option 2). The format of options are as follows:



Type

A two octet field, denoting the option. Values include:

- 1 - GSS Message Length
- 2 - Nonce Payload

Length

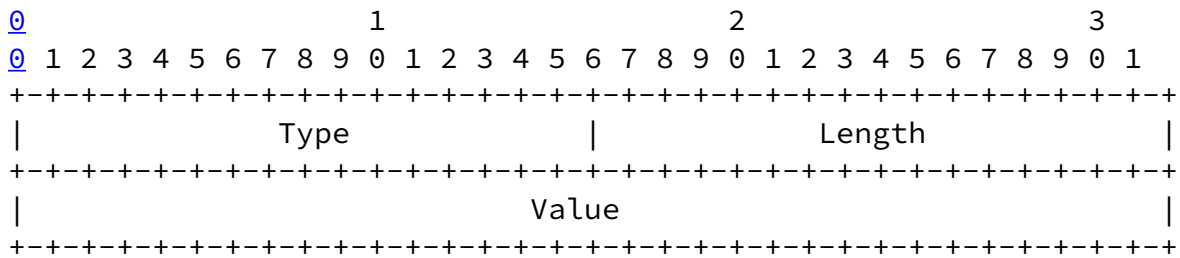
The length of the option, including the type, length and value fields.

Value

The value of the option.

[3.4.1.](#) GSS Message Length option

The GSS Message Length option is present only if the L and O bits are set. It is defined as follows:



Type

1

Length

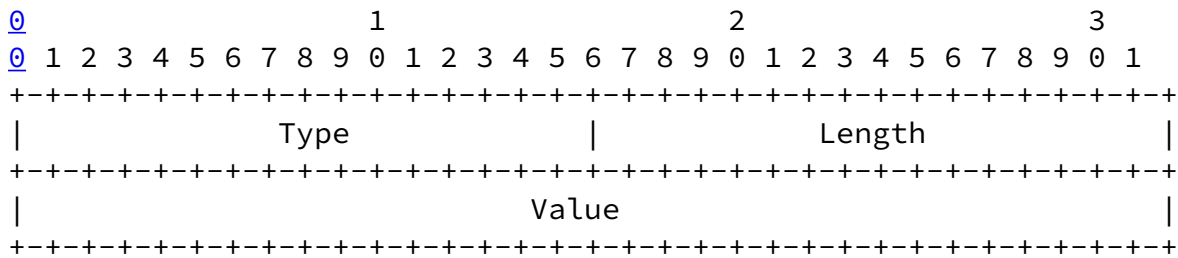
8

Value

The value field of the GSS Message Length options is four octets in length. This field provides the total length of the GSS message or set of messages that is being fragmented.

[3.4.2.](#) Nonce Payload option

The Nonce Payload option is defined as follows:



Type

2

Length

36

Value

The value field of the Nonce Payload option is 32 octets in length. As in [RFC2246] the first 4 octets of this field are the time of day when the message was generated (in seconds since the Unix epoch, 12:00 midnight, January 1, 1970 GMT) and the other 28 octets are randomly generated.

3.5. Fragmentation

It is possible that EAP GSS messages may exceed the link MTU size, the maximum RADIUS packet size of 4096 octets, or even the PPP Multilink Maximum Received Reconstructed Unit (MRRU). As described in [RFC1990], within PPP the multi-link MRRU is negotiated via the Multilink MRRU LCP option, which includes an MRRU length field of two octets, and thus can support MRRUs as large as 64 KB.

In order to protect against reassembly lockup and denial of service attacks, it may be desirable for an implementation to set a maximum size for a GSS-API token. Since a typical certificate chain is rarely longer than a few thousand octets, and no other field is likely to be anywhere near as long, a reasonable choice of maximum acceptable message length might be 64 KB.

If this value is chosen, then for PPP links, fragmentation can be handled via the multi-link PPP fragmentation mechanisms described in [RFC1990]. While this is desirable, there may be cases in which multi-link or the MRRU LCP option cannot be negotiated. Also, since EAP

methods must also be usable within IEEE 802.1X [[IEEE8021X](#)], an EAP GSS implementation MUST provide its own support for fragmentation and reassembly.

Since EAP is a simple ACK-NAK protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field as is provided in IP.

EAP GSS fragmentation support is provided through addition of a flags octet within the EAP-Response and EAP-Request packets, as well as a GSS Message Length field of four octets. Flags include the Length included (L), More fragments (M), Options (O) and EAP GSS Start (S) bits. The L is set to indicate the presence of the GSS Message Length option, and MUST be set for the first fragment of a fragmented GSS message or set of messages. The M flag is set on all but the last fragment. The S flag is set only within the EAP GSS start message sent from the EAP server to the peer. The GSS Message Length option provides the total length of the GSS-API token or set of messages that is being fragmented; this simplifies buffer allocation.

When an EAP GSS peer receives an EAP-Request packet with the M bit set, it MUST respond with an EAP-Response with EAP-Type=EAP GSS and no data. This serves as a fragment ACK. The EAP server MUST wait until it receives the EAP-Response before sending another fragment. In order to prevent errors in processing of fragments, the EAP server MUST increment the Identifier field for each fragment contained within an EAP-Request, and the peer MUST include this Identifier value in the fragment ACK contained within the EAP-Response. Retransmitted fragments will contain the same Identifier value.

Similarly, when the EAP server receives an EAP-Response with the M bit set, it MUST respond with an EAP-Request with EAP-Type=EAP GSS and no data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors in the processing of fragments, the EAP server MUST use increment the Identifier value for each fragment ACK contained within an EAP-Request, and the peer MUST include this Identifier value in the subsequent fragment contained within an EAP-Response.

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

In the case where the EAP GSS authentication is successful, and fragmentation is required, the conversation will appear as follows:

Authenticating Peer	NAS
-----	-----
	EAP-Request/ <- Identity
EAP-Response/ Identity (MyID) ->	
	EAP-Request/ EAP-Type=EAP GSS <- GSS Start, 0 and S bit set, Nonce Payload
GSS_Init_sec_context(mutual_req_flag) returns GSS_S_CONTINUE_NEEDED, output_token (SPNEGO)	
EAP-Response/ EAP-Type=EAP GSS 0 bit set, Nonce Payload, output_token ->	
	GSS_Accept_sec_context(input_token) returns GSS_S_COMPLETE, output_token (SPNEGO)
	EAP-Request/ EAP-Type=EAP GSS output_token <- (Fragment 1: L, 0, M bits set)
EAP-Response/ EAP-Type=EAP GSS ->	
	EAP-Request/ EAP-Type=EAP GSS <- (Fragment 2: M bit set)
EAP-Response/ EAP-Type=EAP GSS ->	
	EAP-Request/

```
EAP-Type=EAP GSS
<- (Fragment 3)
```

```
GSS_Init_sec_context(input_token)
  returns GSS_S_COMPLETE,
  output_token
```

```
EAP-Response/
```

Aboba

Experimental

[Page 17]

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

```
EAP-Type=EAP GSS
output_token
(Fragment 1:
  L, 0, M bits set)->
```

```
EAP-Request/
<- EAP-Type=EAP GSS
```

```
EAP-Response/
EAP-Type=EAP GSS
(Fragment 2)->
```

```
<- EAP-Success
```

[3.6.](#) Retry behavior

As with other EAP protocols, the EAP server is responsible for retry behavior. This means that if the EAP server does not receive a reply from the peer, it MUST resend the EAP-Request for which it has not yet received an EAP-Response. However, the peer MUST NOT resend EAP-Response packets without first being prompted by the EAP server.

For example, if the initial EAP GSS start packet sent by the EAP server were to be lost, then the peer would not receive this packet, and would not respond to it. As a result, the EAP GSS start packet would be resent by the EAP server. Once the peer received the EAP GSS start packet, it would send an EAP-Response encapsulating the client_hello message. If the EAP-Response were to be lost, then the EAP server would resend the initial EAP GSS start, and the peer would resend the EAP-Response.

As a result, it is possible that a peer will receive duplicate EAP-Request messages, and may send duplicate EAP-Responses. Both the peer and the EAP-Server should be engineered to handle this possibility.

[3.7.](#) Identity verification

As part of the GSS-API conversation, it is possible that the server may present a certificate to the peer, or that the peer may present a certificate to the EAP server. If the peer has made a claim of identity in the EAP-Response/Identity (MyID) packet, the EAP server SHOULD verify that the claimed identity corresponds to the certificate presented by the peer. Typically this will be accomplished either by placing the `userId` within the peer certificate, or by providing a mapping between the peer certificate and the `userId` using a directory service.

Similarly, the peer MUST verify the validity of the EAP server certificate, and SHOULD also examine the EAP server name presented in the certificate, in order to determine whether the EAP server can be trusted. Please note that in the case where the EAP authentication is remotated that the EAP server will not reside on the same machine as the

NAS, and therefore the name in the EAP server's certificate cannot be expected to match that of the intended destination. In this case, a more appropriate test might be whether the EAP server's certificate is signed by a CA controlling the intended destination and whether the EAP server exists within a target sub-domain.

[3.8.](#) Use of addresses

When using EAP GSS, the EAP client may not be able to include an address in an EAP-Response message, since prior to obtaining access the EAP client may not have an IP address. This limits effective use of EAP GSS to GSS-API methods that do not require the peer to have an IP address prior to authentication.

The IAKERB GSS-API method can explicitly handle this situation, as described in [[IAKERB](#)]. However, where the Kerberos V protocol, described in [[RFC1510](#)], is negotiated as a GSS-API method as described in [[RFC1964](#)], the `addresses` field of the `AS_REQ` and `TGS_REQ` SHOULD be blank and the `caddr` field of the ticket SHOULD also be left blank.

[4.](#) Key management

As a result of the EAP GSS conversation, the EAP endpoints will mutually authenticate and derive a master key. The master key is then used to derive master session keys for authentication and encryption as well as initialization vectors in each direction. It is the master session keys

that are provided by the EAP method hosted on the client and AAA server, and communicated by the AAA server to the NAS.

The master session keys are then used by the client and NAS in order to derive ciphersuite-specific keys, once the negotiated ciphersuite is known. Depending on the negotiated ciphersuite, not all of the master session keys will be used in this process.

By requiring master session keys (but not ciphersuite-specific keys) to be derived by the EAP method, it is possible for EAP methods to derive keying material that can be used by any ciphersuite. This is desirable, since it avoids having to revise EAP methods each time a new ciphersuite is deployed for any of the applications using EAP.

[4.1.](#) Key derivation algorithm

EAP TLS, described in [[RFC2716](#)], provides a mechanism for deriving authentication and encryption keys as well as IVs in both directions from the TLS master key. The key hierarchy of EAP GSS is similar to that of EAP TLS, with the following differences:

- [1] Pseudo-master key derivation. TLS APIs typically do not enable direct access to the master key, for security reasons. As a result, EAP TLS utilizes the TLS PRF in the master session key derivation, rather than acting on the master key itself.

For similar reasons, EAP GSS implementations will typically not be able to obtain access to the master key via GSS-API. However, GSS-API methods can call GSS_Wrap() to encrypt and GSS_GetMIC() to generate authentication tokens based on the master key. Since EAP GSS cannot assume direct access to the master key, it is not possible to utilize the master key directly in the derivation of the master session keys.

Instead, an intermediate step is required, where a "Pseudo-Master Key" K' is derived from the master key, and used in the derivation of the master session keys. Note that since the randomness of the GSS-API MIC cannot be guaranteed, it is preferable to use GSS_Wrap() to generate the K' , and additional randomness needs to be introduced into the derivation to ensure sufficient entropy.

- [2] Nonce generation. TLS includes an exchange of nonces, and the exchanged nonces are used within the keying hierarchy in order to ensure liveness in the derived master session keys. GSS-API methods such as Kerberos do not include such a nonce exchange, although typically some other source of "liveness" is provided, such as a counter or a time value. The variation in GSS-API methods makes it difficult to come up with a key hierarchy that can be used with any GSS-API method, if only GSS-API calls are available. To circumvent this limitation, EAP GSS adds a nonce exchange patterned after [\[RFC2246\]](#).

In the techniques described here, master session keys are derived from the master key derived by the GSS-API method, but are never used to encrypt or decrypt data; they are only used in the derivation of transient session keys.

The derivation proceeds as follows:

- [1] The "Pseudo-Master Key" K' is derived from the raw master key using the formula: $K' = \text{HMAC-SHA1}("", \text{GSS_Wrap}(\text{"EAP GSS pseudo master key"}) \parallel \text{random})$. Here random is defined as the concatenation of the message fields client_nonce and server_nonce (in that order).
- [2] Given the K' value and the pseudorandom function (PRF) defined in [Appendix B](#), the value $\text{PRF1} = \text{PRF}(K', \text{"client EAP encryption"}, \text{random})$ is computed up to 128 bytes, and the $\text{PRF2} = \text{PRF}("", \text{"client EAP encryption"}, \text{random})$ is computed up to 64 bytes (where "" is an empty string). Here random is defined as the concatenation of the

message fields client_nonce and server_nonce (in that order).

- [3] The peer encryption key (the one used for encrypting data from peer to authenticator) is obtained by truncating to the correct length the first 32 bytes of PRF1. The authenticator encryption key (the one used for encrypting data from authenticator to peer), if different from the client encryption key, is obtained by truncating to the correct length the second 32 bytes of PRF1. The peer authentication key (the one used for computing MICs for messages from peer to authenticator), if used, is obtained by truncating to the correct length the third 32 bytes of PRF1. The authenticator authentication key (the one used for computing MICs for messages

from authenticator to peer), if used, and if different from the peer authentication key, is obtained by truncating to the correct length the fourth 32 bytes of PRF1. The peer initialization vector (IV), used for messages from peer to authenticator if a block cipher has been specified, is obtained by truncating to the cipher's block size the first 32 bytes of PRF2. Finally, the authenticator initialization vector (IV), used for messages from peer to authenticator if a block cipher has been specified, is obtained by truncating to the cipher's block size the second 32 bytes of PRF2.

The use of these encryption, authentication keys and IVs is specific to the ciphersuite used. Additional keys or other non-secret values (such as IVs) can be obtained as needed for future ciphersuites by extending the outputs of the PRF beyond 128 bytes and 64 bytes, respectively.

A description of the key hierarchy is provided on the next page.

[5. Security Considerations](#)

[5.1. Dictionary attacks](#)

As noted in [[KRBATTACK](#)],[[KERBLIM](#)],[[KERB4WEAK](#)], both Kerberos IV and V are vulnerable to dictionary attack. These attacks are particularly potent when carried out in a location where a large number of authentication exchanges can be collected within a short period of time, such as with wireless LANs deployed in "hot spots".

As noted in [[KRBATTACK](#)], offline dictionary attacks are easily carried out against the AS_REP, since the key encrypting the enclosed Kerberos ticket is a function of the password. Such attacks are amenable to parallelization, and it is therefore possible to crack a large number of passwords in short time with only modest resources. The imposition of a password policy is likely only to decrease the yield, but given access to sufficient exchanges, large scale password compromise remains possible.

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|
|      Derivation of the pseudo
|      master key from the
|      raw master key
|
```

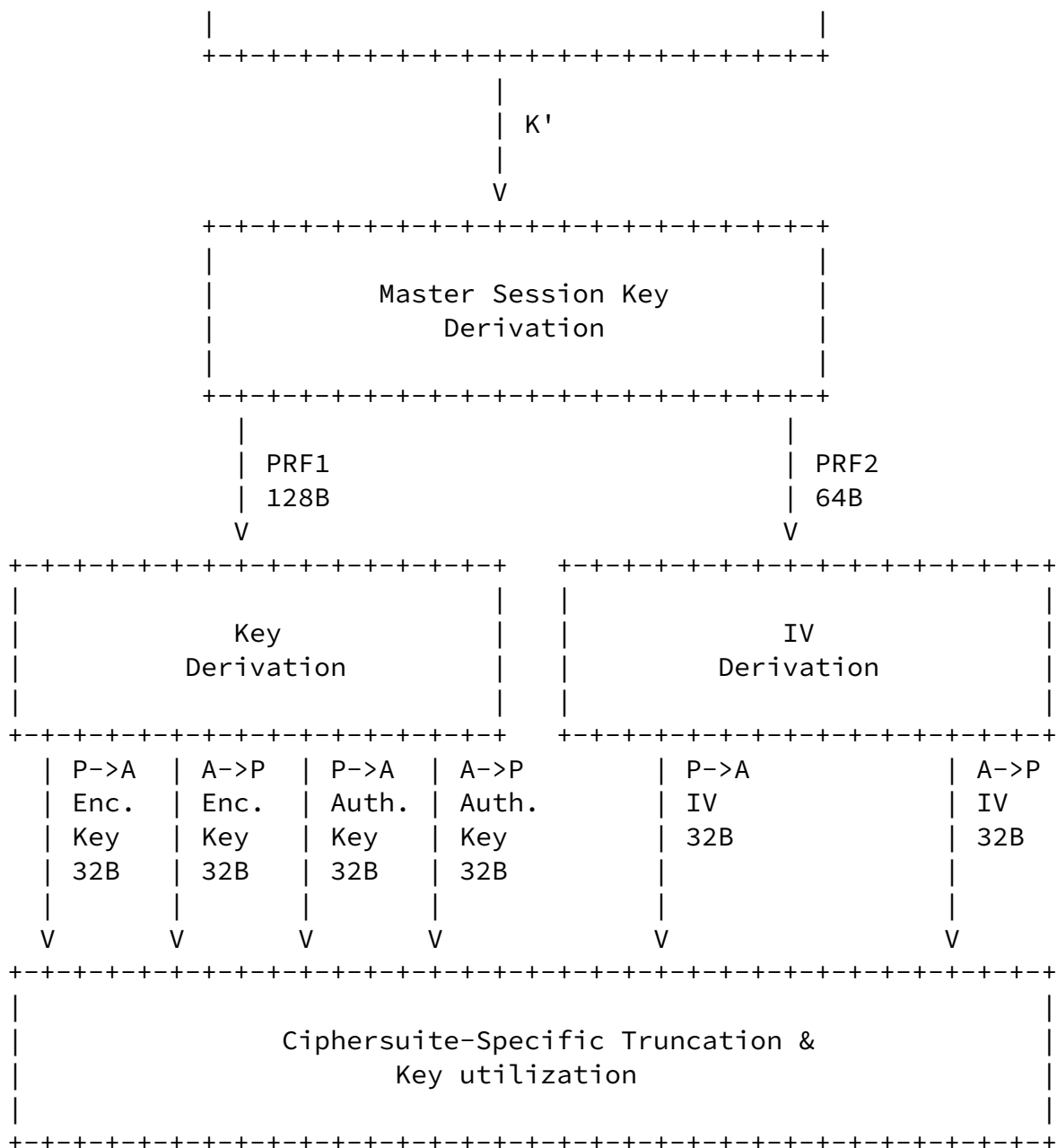



Figure 1 - Algorithm for derivation of session keys from the GSS-API method master key K.

For this reason, when used on wireless networks, EAP GSS SHOULD negotiate methods invulnerable to offline dictionary attacks. This includes public key authentication techniques such as [\[PKINIT\]](#), or password-based techniques such as SRP, described in [\[RFC2945\]](#), EKE, described in [\[EKE\]](#), or techniques described in [\[STRONGAUTH\]](#) or [\[DUAL\]](#).

Kerberos V SHOULD NOT be used without extensions providing protection against offline dictionary attacks. As noted in [\[KRBATTACK\]](#), it has been proposed that Kerberos V dictionary attack vulnerabilities be addressed via a pre-authentication exchange. The vulnerability can also be addressed by use of public key authentication with Kerberos, described in [\[PKINIT\]](#).

[5.2.](#) Certificate revocation

Since the EAP server is typically connected to the Internet during the EAP conversation, it is capable of following a certificate chain or verifying whether the peer's certificate has been revoked. In contrast, the peer may or may not have Internet connectivity, and thus while it can validate the EAP server's certificate based on a pre-configured set of CAs, it may not be able to follow a certificate chain or verify whether the EAP server's certificate has been revoked.

In the case where the peer is initiating a voluntary Layer 2 tunnel using PPTP or L2TP, the peer will typically already have a PPP interface and Internet connectivity established at the time of tunnel initiation. As a result, during the EAP conversation it is capable of checking for certificate revocation.

However, in the case where the peer is initiating a connection, it will not have Internet connectivity and is therefore not capable of checking for certificate revocation until after the peer has access to the Internet. In this case, the peer SHOULD check for certificate revocation after connecting to the Internet.

[5.3.](#) Mutual authentication

In order to guard against rogue NAS devices, it is recommended that a GSS-API method supporting mutual authentication be selected during the SPNEGO negotiation, as described in [\[RFC2478\]](#). This also avoids potential reflection attacks against dual one-way authentication methods, such as EAP-MD5.

[5.4.](#) Credential reuse

A peer with valid credentials may reuse those credentials in a subsequent authentication. Credential reuse improves efficiency in a number of scenarios. Where the peer attempts to re-authenticate to an

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

EAP server within a short period of time, the re-authentication time may be shortened. Also, where the peer roams to another NAS willing to accept credentials from a previous NAS, fast-handoff may be achieved. Credential reuse may also prove useful during multi-link authentication.

For example, a peer initially using the IAKERB GSS-API method to obtain a TGT and a ticket to the NAS may subsequently reuse that ticket in an AP_REQ/AP_REP exchange. Such an exchange may occur either in-band (e.g. via use of the Kerberos V GSS-API method) or out-of-band (e.g. via an 802.1X EAPOL-Key message). Typically in-band efficiency savings are modest (one round-trip saved using the Kerberos V GSS-API method versus IAKERB), since the authentication typically is remoted to the backend authentication server. The savings from out-of-band credential reuse can be more substantial, since in this case the credential validation may occur on the NAS.

The decision of whether to attempt to reuse credentials is left up to the peer, which needs to determine whether credential use is likely to succeed. The decision may be based on out-of-band information (such as probe/response messages exchanged via 802.11 [[IEEE80211](#)]), or the time elapsed since the previous authentication attempt.

If the peer attempts to reuse credentials that are not valid, then it will receive an error response and the peer can re-authenticate using the more complete sequence. For example, after an initial IAKERB authentication, the peer will have obtained a TGT from the KDC via the AS_REP, and a ticket for network access via the TGS_REP. The peer may subsequently attempt to negotiate the Kerberos V GSS-API method, so as to reuse the previously obtained credentials. Should a KRB_ERROR be returned to the peer, then the peer can negotiate IAKERB on its next attempt instead.

Note that credential reuse for the purpose of "fast handoff" has significant limitations. For use in "fast handoff", it is desirable for Kerberos ticket validation to occur on the NAS, rather than remoting the validation to the backend authentication server, since this will save a round-trip between the NAS and the backend authentication server.

However, to enable the peer to reuse a Kerberos ticket on a different NAS, it is necessary for NASen within the same geographic area to share a key with the KDC. If this is not the case, then peers moving from one NAS to another will not be able to reuse credentials without either

requiring communication between the NASen, or remoting of the credential validation to a backend authentication server.

Allowing multiple NASen to share a key with the KDC makes it more likely that an attacker sniffing the wire will be able to obtain the NAS key, particularly if the key is derived from a password. Details are provided

within reference [[KRBATTACK](#)]. Alternatively, the new NAS can pass the submitted ticket and authenticator to a backend authentication server or to the previous NAS for validation. However, requiring communication between the NAS and backend authentication server for "fast handoff" adds substantial to the delay. In addition if it is assumed that the NASen support an Inter-Access Point Protocol (IAPP), then EAP-based "fast handoff" is not necessary at all.

Similarly, if the EAP servers are set up in a rotary or made available via a round-robin technique, then the credentials also may not be reusable without communication with a backend authentication server or previous NAS.

Furthermore, since existing Kerberos implementations do not include AAA authorizations within the authorization data field of the Kerberos ticket [[RFC1510](#)], even if the credentials can be reused, it may be necessary for the NAS to obtain the authorization information from the AAA server before the correct session state can be re-established on the new NAS. If AAA authorizations are not obtained prior to granting access, then the new NAS could potentially provide the wrong service to the peer. For example, where Filter-Id [[RFC2865](#)] or tunnel attributes [[RFC2868](#)] were unavailable, a peer might be given unrestricted network access where this was not intended.

As a result of these considerations, credential reuse for the purpose of "fast handoff" does not appear to be practical at this time.

[5.5](#). Key transmission issues

As a result of the EAP GSS conversation, the EAP endpoints will mutually authenticate and derive a session key, which this specification calls the "master key". Within GSS-API, it is possible to use GSS_Wrap() to encrypt using the master key, or GSS_GetMIC() to produce a message integrity check (MIC) keyed by the master key. However, for security reasons, there are no GSS-API calls to obtain the master key itself.

In the most general case, authentication keys, encryption keys and IVs may be required for use with the chosen ciphersuite. The keys from which these session keys are derived are known as "master session keys" and are derived from the master key.

Since the peer and EAP client reside on the same machine, and the master key cannot be exported, it is necessary for the master session keys to be derived within EAP GSS. Once the ciphersuite has been determined, the master session keys are converted to ciphersuite-specific session keys and passed to the link layer encryption module.

The situation may be more complex on the NAS, which may or may not reside on the same machine as the EAP server. In the case where the EAP server and NAS reside on different machines, there are several implications for security. Firstly, the mutual authentication defined in EAP GSS will occur between the peer and the EAP server, not between the peer and the NAS.

This means that as a result of the EAP GSS conversation, it is not possible for the peer to validate the identity of the device that it is speaking to. The second issue is that the master session keys negotiated between the peer and EAP server will need to be transmitted to the NAS.

Both issues can be addressed via addition of a followon exchange. For example, where the IAKERB GSS-API method is used for initial authentication, the Kerberos V GSS-API method can be used to mutually authenticate the peer and NAS and transfer the master session key from the peer to the NAS, enclosed in a Kerberos ticket. The NAS can then derive the master session keys from the master key. Once the ciphersuite is determined, the master session keys are converted to ciphersuite-specific session keys and passed to the link layer encryption module on the NAS.

5.6. Protected negotiation

SPNEGO [[RFC2478](#)] supports protected method negotiation in the case where the negotiated method provides authentication and integrity protection. In contrast, EAP, described in [[RFC 2284](#)], does not provide for protected method negotiation.

Link layer ciphersuite negotiations are also typically unprotected. For example, ECP, described in [[RFC1968](#)], supports unprotected cipher-suite negotiations within PPP and is thus vulnerable to attack. Similarly, 802.11, described in [[IEEE80211](#)], does not support protected ciphersuite negotiations.

Since peers completing the GSS-API SPNEGO negotiation will typically implicitly select a ciphersuite, which includes key strength, encryption and hashing methods, it is tempting to use this protected ciphersuite negotiation in place of unprotected ciphersuite negotiation mechanisms.

However, use of EAP GSS for protected ciphersuite negotiation presents substantial difficulties, since available link layer ciphersuites may not correspond to the ciphersuites implicitly negotiated as part of SPNEGO.

For example, 802.11 [[IEEE80211](#)] supports Wired Equivalency Privacy (WEP), a flawed cipher based on RC4 which supports confidentiality but lacks a keyed message integrity check. As a result, WEP does not

support per-frame authentication and integrity protection. Similarly, PPP encryption methods such as DESEbis [[RFC2419](#)] and 3DES [[RFC2420](#)] support confidentiality but also do not support per-frame authentication or integrity protection. Since GSS-API ciphersuites available within GSS-API methods such as Kerberos V [[RFC 1964](#)] provide confidentiality as well as per-packet integrity protection and authentication, they do not correspond well to these link layer ciphersuites. As a result, the use of SPNEGO for link-layer ciphersuite negotiation is not recommended.

[6](#). Normative References

[RFC1510] Kohl, J., Neuman, C., "The Kerberos Network Authentication Service (V5)", [RFC 1510](#), September 1993.

[RFC1661] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)." STD 51, [RFC 1661](#), July 1994.

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

[RFC1968] Meyer, G., "The PPP Encryption Protocol (ECP)." [RFC 1968](#), June

1996.

- [RFC1990] Sklower, K., Lloyd, B., McGregor, G., Carr, D., and T. Coradetti, "The PPP Multilink Protocol (MP)." [RFC 1990](#), August 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2246] Dierks, T. and Allen, C. "The TLS Protocol Version 1.0", [RFC 2246](#), November 1998.
- [RFC2284] Blunk, L., Vollbrecht, J., "PPP Extensible Authentication Protocol (EAP)", [RFC 2284](#), March 1998.
- [RFC2478] Baize, E., Pinkas., D., "The Simple and Protected GSS-API Negotiation Mechanism", [RFC 2478](#), December 1998.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface, Version 2", [RFC 2743](#), January 2000.
- [IEEE8021X]
IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std 802.1X-2001, June 2001.
- [KERB] Neuman, B. C., Ts'o, T., "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, 32(9):33-38,

Aboba

Experimental

[Page 27]

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

September 1994.

- [IAKERB] Swift, M., Trostle, J., Aboba, B., Zorn, G., "Initial Authentication and Pass Through Authentication Using Kerberos V5 and the GSS-API (IAKERB)", Internet draft (work in progress), [draft-ietf-cat-iakerb-08.txt](#), August 2001.

[7.](#) Informative References

- [RFC2104] Krawczyk, H. et al, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [RFC2419] Sklower, K., Meyer, G., "The PPP DES Encryption Protocol, Version 2 (DESE-bis)", [RFC 2419](#), September 1998.

- [RFC2420] Hummert, K., "The PPP Triple-DES Encryption Protocol (3DESE)", [RFC 2420](#), September 1998.
- [RFC2716] Aboba, B., Simon, S., "PPP EAP TLS Authentication Protocol", [RFC 2716](#), October 1999.
- [RFC2865] Rigney, C., Rubens, A., Simpson, W., Willens, S., "Remote Authentication Dial In User Service (RADIUS)", [RFC 2865](#), June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", [RFC 2866](#), June 2000.
- [RFC2867] Zorn, G., Mitton, D., Aboba, B., "RADIUS Accounting Modifications for Tunnel Protocol Support", [RFC 2867](#), June 2000.
- [RFC2868] Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., Goyret, I., "RADIUS Attributes for Tunnel Protocol Support", [RFC 2868](#), June 2000.
- [RFC2869] Rigney, C., Willats, W., Calhoun, P., "RADIUS Extensions", [RFC 2869](#), June 2000.
- [RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", [RFC 2945](#), September 2000.
- [RFC3078] Pall, G. and Zorn, G. "Microsoft Point-to-Point Encryption (MPPE)" [RFC 3078](#), March 2001.
- [RFC3079] Zorn, G. "Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE)," [RFC 3079](#), March 2001.

[IEEE80211]

Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-1997, 1997.

- [IEEE802] IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture, ANSI/IEEE Std 802, 1990.
- [DESFIPS] National Bureau of Standards, "DES Modes of Operation", FIPS PUB 81 (December 1980).
- [KRBATTACK]
Wu, T., "A Real-World Analysis of Kerberos Password Security", Stanford University Computer Science Department,
<http://theory.stanford.edu/~tjw/krbpass.html>
- [KRBLIM] Bellovin, S.M., Merritt, M., "Limitations of the kerberos authentication system", Proceedings of the 1991 Winter USENIX Conference, pp. 253-267, 1991.
- [KERB4WEAK]
Dole, B., Lodin, S., and Spafford, E., "Misplaced trust: Kerberos 4 session keys", Proceedings of the Internet Society Network and Distributed System Security Symposium, pp. 60-70, March 1997.
- [EKE] Bellovin, S.M., Merritt, M., "Encrypted key exchange: Password-based protocols secure against dictionary attacks", Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy, pp. 72-84, 1992.
- [STRONGAUTH]
Jablon, D., "Strong password-only authenticated key exchange", Computer Communication Review, 26(5):5-26, October 1996.
- [DUAL] Jaspan, B., "Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks", Proceedings of the Sixth Annual USENIX Security Conference, pp. 43-50, July 1996.
- [IEEE80211i]
IEEE Draft 802.11i/D2, "Draft Supplement to STANDARD FOR Telecommunications and Information Exchange between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Specification for Enhanced Security", July 2001.

- [AESProp] Daemen, J., Rijman, V., "AES Proposal: Rijndael," NIST AES Proposal, June 1998.
<http://csrc.nist.gov/encryption/aes/round2/AESAlgs/Rijndael/Rijndael.pdf>
- [AESFIPS] Draft FIPS Publication ZZZZ, "Advanced Encryption Standard (AES)", U.S. DoC/NIST, summer 2001.
- [MODES] "Symmetric Key Block Cipher Modes of Operation,"
<http://www.nist.gov/modes>.
- [BLOCK] "Recommendation for Block Cipher Modes of Operation", National Institute of Standards and Technology (NIST) Special Publication 800-XX, CODEN: NSPUE2, U.S. Government Printing Office, Washington, DC, July 2001.
- [PKINIT] Tung, B. Neuman, C., Hur, M., Medvinsky, A., Medvinsky, S., Wray, J., Trostle, J., "Public Key Cryptography for Initial Authentication in Kerberos", [draft-ietf-cat-kerberos-pk-init-13.txt](#), August 2001.

IANA Considerations

This document requires assignment of a EAP Type for EAP GSS. It does not create any new number spaces for IANA administration.

Acknowledgments

Thanks to Paul Leach of Microsoft, Glen Zorn of Cisco Systems, and Jesse Walker of Intel for useful discussions of this problem space.

Authors' Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Phone: +1 425 706 6605
EMail: bernarda@microsoft.com

Dan Simon
Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: dansimon@microsoft.com

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

Phone: +1 425 706 6711

Appendix A - Example IAKERB topologies

Where EAP GSS is used along with the GSS-API IAKERB [[IAKERB](#)] or Kerberos V [[RFC1964](#)] mechanisms, two major topologies are possible:

RADIUS+KDC backend

Here a RADIUS backend is used, along with a Kerberos KDC. The NAS functions as an EAP-pass-through device as described in [[RFC2284](#)]. This involves encapsulating EAP messages received from the peer within RADIUS as described in [[RFC2869](#)], and passing them on to the RADIUS server. In turn, the RADIUS server acts as an IAKERB proxy, de-capsulating EAP GSS/IAKERB packets, and passing them on to the Kerberos KDC. In turn, the RADIUS server will encapsulate packets from the Kerberos KDC in EAP GSS/IAKERB and send them to the NAS. EAP-Message attributes received from the RADIUS server are de-capsulated by the NAS and sent to the peer. In this topology, the NAS need not have knowledge of specific EAP or GSS-API methods, while the RADIUS server does require this knowledge.

KDC backend

In this topology, only a Kerberos KDC is used as a backend, and the NAS functions as an IAKERB proxy, de-capsulating EAP GSS/IAKERB messages and passing them on to the KDC. Messages from the KDC are encapsulated within EAP GSS/IAKERB by the NAS and sent to the peer. In this case, the NAS needs to understand the EAP GSS, GSS-API IAKERB, as well as GSS-API Kerberos V mechanisms. In addition, where the peer already has a valid TGT and ticket to the NAS, it may choose to use the Kerberos V mechanism within EAP. Note that in the case of 802.11, the Kerberos AP_REQ/AP_REP messages may be carried in messages outside the conventional EAP exchange, such as those defined in [[IEEE8021X](#)] so that use of the Kerberos V mechanism within EAP is not necessary.

In the examples below, each topology is discussed. While nominally the EAP conversation occurs between the NAS and the peer, the NAS MAY act as a pass-through device, with the EAP packets received from the peer being encapsulated for transmission to a RADIUS server. In the discussion that follows, we will use the term "EAP server" to denote the ultimate endpoint conversing with the peer.

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

[A.1](#) RADIUS+KDC backend

In this topology, the NAS will act as an EAP pass-through, and the RADIUS server acts as an IAKERB proxy. A successful EAP GSS/IAKERB authentication will appear as follows:

Peer	NAS	RADIUS	KDC
-----	-----	-----	-----
	EAP/Identity <-Request		
EAP/Identity Response ->			
	EAP/Identity Response ->		
		Access-Challenge EAP GSS Request <- (Start)	
	<-EAP GSS Request(Empty)		
EAP GSS Response [1] (SPNEGO) ->			
	EAP GSS Response (SPNEGO) ->		
		Access-Challenge EAP GSS Request <-(SPNEGO)	
	EAP GSS Request <-(SPNEGO)		
EAP GSS IAKERB			

Response
(Empty) ->

Access-Accept [4]
<- EAP-Success

<- EAP-Success

AP_REQ ->

<- AP_REP [5]

Notes:

- [1.](#) IAKERB may be requested by the EAP GSS client without the need for negotiation, or SPNEGO may be used.
- [2.](#) The AS_REQ requests a TGT from the KDC. It may or may not include PADATA. As a result, the AS_REQ may not authenticate the peer to the KDC, but the AS_REP authenticates the KDC to the peer.

Aboba

Experimental

[Page 33]

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

- [3.](#) The TGS_REQ requests a ticket to the NAS service. The ticket is encrypted with the NAS's key so that it can only be validated by the NAS.
- [4.](#) On receiving a TGS_REP from the KDC rather than a KRB_ERROR, the RADIUS server can conclude that the peer has successfully authenticated, and thus that it is appropriate to reply to the NAS with an Access-Accept encapsulating an EAP-Success.
- [5.](#) The IAKERB exchange ends before the AP_REQ/AP_REP exchange occurs. As a result, the AP_REQ/AP_REP exchange either will not occur (preventing mutual authentication between peer and NAS or transport of the session key from peer to NAS), will occur out-of-band (e.g. after access is granted), or will occur in a subsequent EAP GSS conversation (e.g. using the GSS-API Kerberos V method).

[A.2](#) Kerberos KDC backend

In this topology, there is no RADIUS server, and the NAS functions as an IAKERB proxy, de-capsulating EAP GSS/IAKERB frames and passing them on to the KDC. In turn, packets from the KDC are encapsulated in EAP GSS/IAKERB frames and sent to the peer by the NAS. Where IAKERB is

used, the NAS functions as an IAKERB proxy, de-capsulating EAP GSS/IAKERB messages and passing them on to the KDC. In addition, where the peer already has a valid TGT and ticket to the NAS, it may choose to use the Kerberos V mechanism within EAP. Note that in the case of 802.11, the Kerberos AP_REQ/AP_REP messages are carried in messages outside the conventional EAP exchange, such as the EAPOL-Key messages described in [[IEEE8021X](#)] so that use of the Kerberos V mechanism within EAP is not necessary.

In the Kerberos-only topology, messages from the KDC are encapsulated within EAP GSS/IAKERB and sent to the peer. In this case, the NAS needs to understand the EAP GSS, GSS-API IAKERB, as well as GSS-API Kerberos V mechanisms.

A successful EAP GSS/IAKERB authentication occurring in a topology with a NAS acting as an IAKERB proxy to a Kerberos KDC will appear as follows:

Peer	NAS	KDC
-----	-----	-----
	EAP/Identity	
	<-Request	
EAP/Identity		
Response ->		
	<-EAP GSS Start	
EAP GSS IAKERB		

Response [1]
(AS_REQ) ->

AS_REQ ->

<- AS_REP [2]

EAP GSS IAKERB Request
<-AS_REP)

EAP GSS IAKERB
Response [3]
(TGS_REQ) ->

TGS_REQ ->

<- TGS_REP [4]

EAP GSS IAKERB Request
<-(TGS_REP)

EAP GSS IAKERB
Response
(Empty) ->

<- EAP-Success

AP_REQ [5]->

<- AP_REP [6]

Notes:

1. If PADATA is not used in the AS_REQ, then the peer does not authenticate to the KDC.

2. The KDC authenticates to the peer in the AS_REP.
3. The peer authenticates to the KDC via the TGS_REQ.
4. The KDC authenticates to the peer via the TGS_REP. The TGS_REP also provides the peer with a ticket and session-key for use with the NAS.

5. Up until this point, the peer has not mutually authenticated with the NAS, or exchanged a key with it. As a result, the peer and NAS need to conclude an AP_REQ/AP_REP exchange. This can occur in-band or out-of-band. In the AP-REQ, the peer authenticates to the NAS and provides it with a session key.
6. The NAS authenticates to the peer using the AP_REP.

Appendix B - The Pseudorandom Function

Given below is the description of the HMAC and pseudorandom function based on [Section 5](#) of the TLS Protocol Version 1.0 specification [\[RFC2246\]](#).

Some of operations in the key derivation require a keyed MIC; this is a secure digest of some data protected by a secret. Forging the MIC is infeasible without knowledge of the MIC secret. The construction we use for this operation is known as HMAC, described in [\[RFC2104\]](#).

HMAC can be used with a variety of different hash algorithms. We use it with two different algorithms: MD5 and SHA-1, denoting these as HMAC_MD5(secret, data) and HMAC_SHA1(secret, data). Additional hash algorithms can be defined and used to protect record data, but MD5 and SHA-1 are hard coded into the protocol.

In addition, a construction is required to do expansion of secrets into blocks of data for the purposes of key generation or validation. This pseudo-random function (PRF) takes as input a secret, a seed, and an identifying label and produces an output of arbitrary length.

In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure.

First, we define a data expansion function, P_hash(secret, data) which uses a single hash function to expand a secret and seed into an arbitrary quantity of output:

$$\text{P_hash}(\text{secret}, \text{seed}) = \text{HMAC_hash}(\text{secret}, A(1) + \text{seed}) + \\ \text{HMAC_hash}(\text{secret}, A(2) + \text{seed}) +$$
$$\text{HMAC_hash}(\text{secret}, A(3) + \text{seed}) + \dots$$

Where + indicates concatenation. A() is defined as:

```
A(0) = seed  
A(i) = HMAC_hash(secret, A(i-1))
```

P_hash can be iterated as many times as is necessary to produce the required quantity of data. For example, if P_SHA-1 was being used to create 64 bytes of data, it would have to be iterated 4 times (through A(4)), creating 80 bytes of output data; the last 16 bytes of the final iteration would then be discarded, leaving 64 bytes of output data.

The PRF is created by splitting the secret into two halves and using one half to generate data with P_MD5 and the other half to generate data with P_SHA-1, then exclusive-or'ing the outputs of these two expansion functions together.

S1 and S2 are the two halves of the secret and each is the same length. S1 is taken from the first half of the secret, S2 from the second half. Their length is created by rounding up the length of the overall secret divided by two; thus, if the original secret is an odd number of bytes long, the last byte of S1 will be the same as the first byte of S2.

```
L_S = length in bytes of secret;  
L_S1 = L_S2 = ceil(L_S / 2);
```

The secret is partitioned into two halves (with the possibility of one shared byte) as described above, S1 taking the first L_S1 bytes and S2 the last L_S2 bytes.

The PRF is then defined as the result of mixing the two pseudorandom streams by exclusive-or'ing them together.

```
PRF(secret, label, seed) = P_MD5(S1, label + seed) XOR  
                           P_SHA-1(S2, label + seed);
```

The label is an ASCII string. It should be included in the exact form it is given without a length byte or trailing null character. For example, the label "slithy toves" would be processed by hashing the following bytes:

[73](#) 6C 69 74 68 79 20 74 6F 76 65 73

Note that because MD5 produces 16 byte outputs and SHA-1 produces 20 byte outputs, the boundaries of their internal iterations will not be aligned; to generate a 80 byte output will involve P_MD5 being iterated through A(5), while P_SHA-1 will only iterate through A(4).

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.

Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.
This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

INTERNET-DRAFT

EAP GSS Authentication Protocol

6 April 2002

Expiration Date

This memo is filed as <[draft-aboba-pppext-eapgss-12.txt](#)>, and expires November 23, 2002.

