EAP Generalized Pre-Shared Key (EAP-GPSK)
draft-ietf-emu-eap-gpsk-01.txt

Status of this Memo

Copyright Notice

Abstract

This Internet Draft defines an Extensible Authentication Protocol
method called EAP Generalized Pre-Shared Key (EAP-GPSK). This method
is a lightweight shared-key authentication protocol supporting mutual
authentication and key derivation.

Table of Contents

1. Introduction

   EAP Generalized Pre-Shared Key (EAP-GPSK) is an EAP method defining a
   generalized pre-shared key authentication technique. Mutual
   authentication is achieved through a nonce-based exchange that is
   secured by a pre-shared key.

   At present, several pre-shared key EAP methods are specified, most
   notably

   o EAP-PAX [I-D.clancy-eap-pax]
   o EAP-PSK [I-D.bersani-eap-psk]
   o EAP-TLS-PSK [I-D.otto-emu-eap-tls-psk] and
   o EAP-SAKE [I-D.vanderveen-eap-sake].

   Each proposal has its particular benefits but also its particular
   deficiencies. EAP-GPSK is a new EAP method that tries to combine the
   most valuable characteristics of each of these methods and therefore
   attempts to address a broad range of usage scenarios.

   The main design goals of EAP-GPSK are

   Simplicity:

      EAP-GPSK should be easy to implement and therefore quickly
      available.

   Wide applicability:

      EAP-GPSK has been designed in a threat model where the attacker
      has full control over the communication channel. This is the EAP
      threat model that is presented in Section 7.1 of [RFC3748]. Thus,
      it is particularly suited for wireless or battery powered devices.

   Efficiency:

      EAP-GPSK does not make use of public key cryptography and fully
      relies of symmetric cryptography. The restriction on symmetric
      cryptographic computations allows for low computational overhead.
      Hence, EAP-GPSK is lightweight and well suited for any type of
      device, especially those with little processing power and memory.

   Flexibility:

      EAP-GPSK offers cryptographic flexibility. At the beginning, the
      EAP server selects a set of cryptographic algorithms and key
      sizes, a so called ciphersuite. The current version of EAP-GPSK
      comprises two ciphersuites, but additional ones can be easily

added.

Extensibility:

   The design of EAP-GPSK allows to securely exchange information
   between the EAP peer and the EAP server using protected data
   fields. These fields might, for example, be used to exchange
   channel binding information or to provide support for identity
   confidentiality.

2. Terminology

In this document, several words are used to signify the requirements
of the specification. These words are often capitalized. The key
words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
"SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document
are to be interpreted as described in [RFC2119].

This section describes the various variables and functions used in
the EAP-GPSK method.

Variables:

PD_Payload_X:

   Data carried within the X-th protected data payload
CSuite_List:

   An octet array listing available ciphersuites (variable length)

CSuite_Sel:

   Ciphersuite selected by the client (1 octet or 7 octets)

ID_Client:

   Client NAI [RFC2486bis]

ID_Server:

   Server identity as an opaque blob.

   KS:

      Integer representing the key size in octets of the selected
      ciphersuite CSuite_Sel

   RAND_Client:

      Random integer generated by the client (256 bits)

   RAND_Server:

      Random integer generated by the server (256 bits)

   Operations:

   A || B:

      Concatenation of octet strings A and B

   ENC_X(Y):

      Encryption of message Y with a symmetric key X, using a defined
      block cipher

   KDF_X(Y):

      Key Derivation Function that generates an arbitrary number of
      octets of output using secret X and seed Y

   length(X):

      Function that returns the length of input X in octets, encoded as
      a 16-bit integer in network byte order

   MAC_X(Y):

      Keyed message authentication code computed over Y with symmetric
      key X

   SEC_X(Y):

      SEC is a function that provides integrity protection based on the
      chosen ciphersuite. The function SEC uses the algorithm defined
      by the selected ciphersuite and applies it to the message content
      Y with key X. As an output the message returns Y concatenated with
      MAC_X(Y).

X[A..B]:

   Notation representing octets A through B of octet array X

The following abbreviations are used for the keying material:

PK:

   Session key generated from the MK and used during protocol
   exchange to encrypt protected data (size defined by ciphersuite)

SK:

   Session key generated from the MK and used during protocol
   exchange to prove knowledge of PSK (size defined by ciphersuite)

EMSK:

   Extended Master Session Key is exported by the EAP method (512
   bits)

MK:

   Master Key between the client and EAP server from which all other
   EAP method session keys are derived (KS octets)

MSK:

   Master Session Key exported by to the EAP method (512 bits)

MID:

   Method ID exported by the EAP method according to the EAP keying
   framework [I-D.ietf-eap-keying] (128 bits)

PSK:

   Long-term key shared between the client and the server (PL octets)

3. Overview

The EAP framework [RFC3748] defines four basic steps that occur
during the execution of an EAP conversation between client and
server. The first phase, discovery, is handled by the underlying
protocol. The authentication phase is defined here. The key

distribution and secure association phases are handled differently
depending on the underlying protocol, and are not discussed in this
document.

```
+--------+                                        +--------+
|        |            EAP-Request/Identity |      |        |
|  EAP   |<-------------------------------------| EAP    |
|  peer  |                                     | server |
|        | EAP-Response/Identity               |        |
|        |----------------------------------->|      |        |
|        |                                     |        |
|        |                    EAP-Request/GPSK-1 |      |        |
|        |<----------------------------------|      |        |
|        |                                     |        |
|        | EAP-Response/GPSK-2                  |        |
|        |----------------------------------->|      |        |
|        |                                     |        |
|        |                    EAP-Request/GPSK-3 |      |        |
|        |<----------------------------------|      |        |
|        |                                     |        |
|        | EAP-Response/GPSK-4                  |        |
|        |----------------------------------->|      |        |
|        |                                     |        |
|        |            EAP-Success or EAP-Failure |      |        |
|        |<----------------------------------|      |        |
+--------+                                        +--------+
```
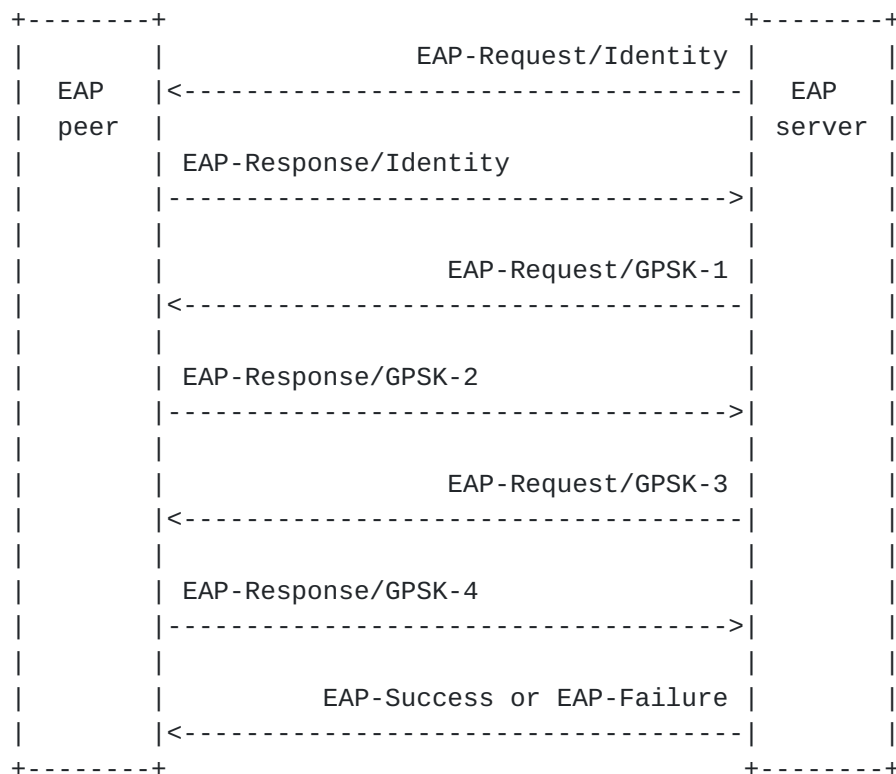
EAP-GPSK performs mutual authentication between EAP peer ("Client")
and EAP server ("Server") based on a pre-shared key (PSK). The
protocol consists of two EAP message exchanges, in which both sides

o   exchange nonces and their identities and

o   compute and exchange a Message Authentication Code (MAC) over the
    previously exchanged values, keyed with the pre-shared key. This
    MAC is considered as proof of possession of the pre-shared key.

The full EAP-GPSK protocol is as follows:
GPSK-1:

    ID_Server, RAND_Server, CSuite_List

GPSK-2:

    SEC_SK(ID_Client, ID_Server, RAND_Client, RAND_Server,
    CSuite_List, CSuite_Sel [, ENC_PK(PD_Payload_1), ...])

```
GPSK-3

   SEC_SK(RAND_Client, RAND_Server, CSuite_Sel [,
   ENC_PK(PD_Payload_2) ] )

GPSK-4:

   [ SEC_SK(ENC_PK(PD_Payload_3)) ]
```

The EAP server begins EAP-GPSK creating a random number RAND_Server
and by encoding the supported ciphersuites into CSuite_List. A
ciphersuite consists of an encryption algorithm, a key derivation
function and a message authentication code.

In GPSK-1, the EAP server sends its identity ID_Server, a random
number RAND_Server and the identifier of the chosen ciphersuite. The
decision which ciphersuite to use is policy-dependent and therefore
outside the scope of this document.

In GPSK-2, the peer sends its identity ID_Client, a random number
RAND_Client. Furthermore, it repeats the received parameters of the
GPSK-1 message and computes a Message Authentication Code over all
these parameters.

The EAP server verifies the received Message Authentication Code. In
case of successful verification, the EAP server computes a Message
Authentication Code over the session parameter and returns it to the
client (within GPSK-3). Within GPSK-2 and GPSK-3, peer and EAP
server have the possibility to exchange encrypted protected data
parameters.

The peer verifies the received Message Authentication Code. If the
verification is successful, GPSK-4 is prepared. This message can
optionally contain the client's protected data parameters.

Upon receipt of GPSK-4, the server assures that the peer has derived
session keys SK and PK properly. Then, the EAP server sends an EAP
Success message to indicate the successful outcome of the
authentication.

4. Key Derivation

EAP-GPSK provides key derivation in compliance to the requirements of
[RFC3748] and [I-D.ietf-eap-keying].

The long-term credential shared between EAP peer and EAP server

SHOULD be a strong pre-shared key PSK of at least 16 bytes, though
its length and entropy is variable. While it is possible to use a
password or passphrase, doing so is NOT RECOMMENDED as it would make
EAP-GPSK vulnerable to dictionary attacks.

During an EAP-GPSK authentication, a Master Key MK, a Session Key SK
and a Protected Data Encryption Key PK are derived using the
ciphersuite-specified KDF and data exchanged during the execution of
the protocol, namely 'RAND_Client || ID_Client || RAND_Server ||
ID_Server' referred as inputString as its short-hand form.

In case of successful completion, EAP-GPSK derives and exports an MSK
and EMSK both in length of 64 bytes. This keying material is derived
using the ciphersuite-specified KDF as follows:

o inputString = RAND_Client || ID_Client || RAND_Server || ID_Server
o MK = KDF_Zero-String (PL || PSK || CSuite_Sel ||
   inputString)[0..KS-1]
o SK = KDF_MK (inputString)[128..127+KS]
o PK = KDF_MK (inputString)[128+KS..127+2*KS]
o MSK = KDF_MK (inputString)[0..63]
o EMSK = KDF_MK (inputString)[64..127]
o MID = KDF_Zero-String ("Method ID" || EAP_Method_Type ||
   CSuite_Sel || inputString)[0..15]

Note that the term 'Zero-String' refers to a sequence of 0x00 values,
KS octets in length. EAP_Method_Type refers to the integer value of
the IANA allocated EAP Type code.

Figure 2 depicts the key derivation procedure of EAP-GPSK.

```
 +-------------+       +-------------------------------+
 |  PL-octet   |       | RAND_Client || ID_Client ||   |
 |    PSK      |       | RAND_Server || ID_Server      |
 +-------------+       +-------------------------------+
        |                            |            |
        v                            v            |
 +---------------------------------------------+  |
 |                   KDF                       |  |
 +---------------------------------------------+  |
                      |                           |
                      v                           |
              +-------------+                     |
              |   KS-octet  |                     |
              |      MK     |                     |
              +-------------+                     |
                      |                           |
                      v                           v
 +-----------------------------------------------------+
 |                        KDF                          |
 +-----------------------------------------------------+
        |           |            |             |
        v           v            v             v
 +---------+   +---------+   +----------+   +----------+
 | 512-bit |   | 512-bit |   | KS-octet |   | KS-octet |
 |   MSK   |   |  EMSK   |   |    SK    |   |    PK    |
 +---------+   +---------+   +----------+   +----------+
```
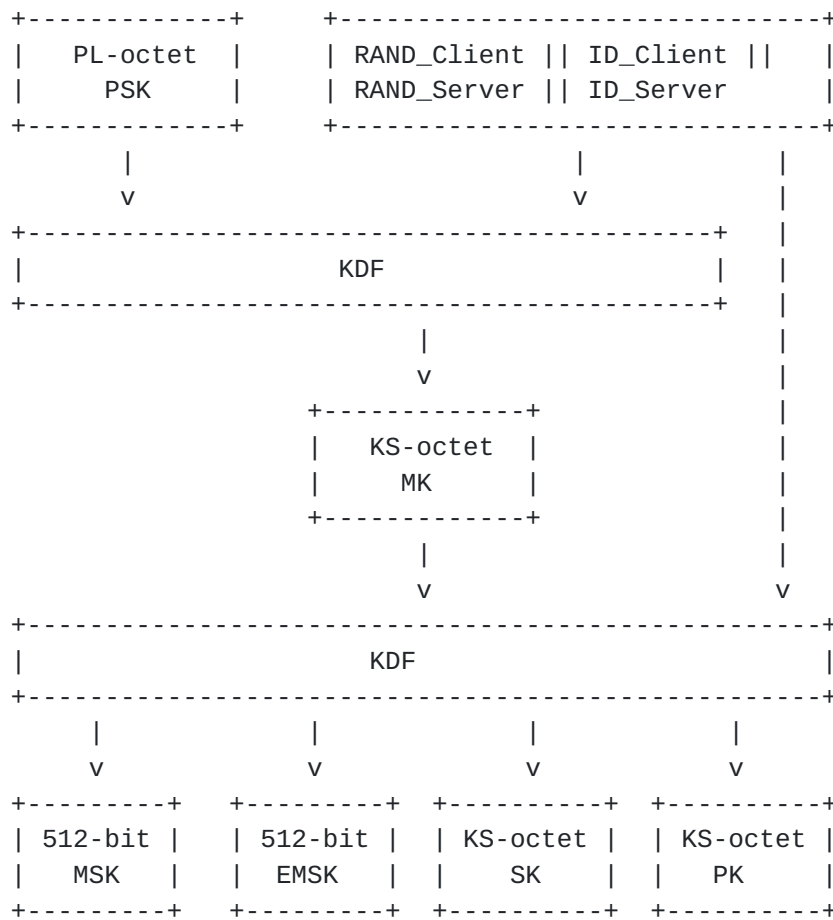
Figure 2: EAP-GPSK Key Derivation

5. Ciphersuites

   The design of EAP-GPSK allows cryptographic algorithms and key sizes,
   called ciphersuites, to be negotiated during the protocol run. The
   ability to specify block-based and hash-based ciphersuites is
   offered. Extensibility is provided with the introduction of new
   ciphersuites; this document specifies an initial set. The CSuite/
   Specifier column in Figure 3 uniquely identifies a ciphersuite.

   For a vendor-specific ciphersuite the first three octets are the
   vendor-specific OID, and the last three octets are vendor assigned
   for the specific ciphersuite.

   The following ciphersuites are specified in this document:

```
+-----------+----+------------+--------------+--------------------+
| CSuite/   | KS | Encryption | Integrity    | Key Derivation     |
| Specifier |    |            |              | Function           |
+-----------+----+------------+--------------+--------------------+
| 0x000001  | 16 | AES-CBC-128| AES_CMAC_128 | GKDF-128           |
+-----------+----+------------+--------------+--------------------+
| 0x000002  | 32 | NULL       | HMAC-SHA256  | GKDF-256           |
+-----------+----+------------+--------------+--------------------+
```

Figure 3: Ciphersuites

Ciphersuite 1, which is based on AES as a cryptographic primitive, is mandatory to implement. This document specifies also a second ciphersuite, but its support is only optional.

Each ciphersuite needs to specify a key derivation function. The ciphersuites defined in this document make use of the Generalized Key Distribution Function (GKDF). Future ciphersuites can use any other formally specified KDF that takes as arguments a key and a seed value, and produces at least 1024+2*KS bits of output.

If GKDF is invoked by a MAC-based ciphersuite, then the variable "size" contains the MAC output size in octets. In case of a block cipher-based ciphersuite, "size" contains the block size in octets.

GKDF has the following structure:

GKDF-X(Y, Z)

X length, in octets, of the desired output
Y secret key used to protect the computation
Z data specific for the protocol run


```
GKDF-X (Y, Z)
{
    n = int( X / size - 1 ) + 1; /* determine number of
                                    output blocks */
    M_0 = "";
    result = "";

    for i=1 to n {
        M_i = MAC_Y (M_{i-1} || Z || i || X);
        result = result || M_i;
    }

    return truncate (result; X)
}
```

   Note that the variables 'i' and 'X' in M_i are represented as 16-bit
   values in network byte order.

6. Ciphersuites Processing Rules

6.1. Ciphersuite #1

6.1.1. Encryption

   With this ciphersuite all cryptography is built around a single
   cryptographic primitive, AES-128. Within the protected data frames,
   AES-128 is used in CBC mode of operation (see [CBC]). The CBC mode
   is well-defined and well-understood. This mode requires an
   Initialization Vector (IV) that has the same size as the block size.
   For security reasons, the IV should be randomly generated.

   In a nutshell, the CBC mode proceeds as follows. The IV is XORed
   with the first plaintext block before it is encrypted. Then for
   successive blocks, the previous ciphertext block is XORed with the
   current plaintext, before it is encrypted.

   Note that in order to provide integrity protection, the CBC-encrypted
   ciphertext MUST be accompanied by a MAC.

   [Editor's Note: Description about the computation of the IV is
   missing]

6.1.2. Integrity

   Ciphersuite 1 uses CMAC as Message Authentication Code. CMAC is
   recommended by NIST. Among its advantages, CMAC is capable to work
   with messages of arbitrary length. A detailed description of CMAC
   can be found in [CMAC].

   The following instantiation is used: AES-128-CMAC(SK, Input) denotes
   the MAC of Input under the key SK.

   where Input refers to the following content:

   o Value of SEC_SK in message GPSK-2
   o Value of SEC_SK in message GPSK-3
   o Value of SEC_SK in message GPSK-4

6.1.3. **Key Derivation**

   This ciphersuite instantiates the KDF in the following way:

   inputString = RAND_Client || ID_Client || RAND_Server || ID_Server

   MK = GKDF-16 (Zero-String, PL || PSK || CSuite_SEL || inputString)

   KDF_out = GKDF-160 (MK, inputString)

   MSK = KDF_out[0..63]

   EMSK = KDF_out[64..127]

   SK = KDF_out[128..143]

   PK = KDF_out[144..159]

   MID = GKDF-16 (Zero-String, "Method ID" || EAP_Method_Type ||
   CSuite_Sel || inputString)

6.2. **Ciphersuite #2**

6.2.1. **Encryption**

   Ciphersuite 2 does not include an algorithm for encryption. With a
   NULL encryption algorithm, encryption is defined as:

   E_X(Y) = Y

   When using this ciphersuite, the data exchanged inside the protected
   data blocks is not encrypted. Therefore this mode MUST NOT be used
   if confidential information appears inside the protected data blocks.

6.2.2. **Integrity**

   Ciphersuite 2 uses the keyed MAC function HMAC, with the SHA256 hash
   algorithm.

   For integrity protection the following instantiation is used:

   HMAC-SHA256(SK, Input) denotes the MAC of Input under the key SK
   where Input refers to the following content:

   o Value of SEC_SK in message GPSK-2
   o Value of SEC_SK in message GPSK-3

o Value of SEC_SK in message GPSK-4

6.2.3. Key Derivation

This ciphersuite instantiates the KDF in the following way:

inputString = RAND_Client || ID_Client || RAND_Server || ID_Server

MK = GKDF-32 (Zero-String, PL || PSK || CSuite_SEL || inputString)

KDF_out = GKDF-192 (MK, inputString)

MSK = KDF_out[0..63]

EMSK = KDF_out[64..127]

SK = KDF_out[128..159]

PK = KDF_out[160..191]

MID = GKDF-16 (Zero-String, "Method ID" || EAP_Method_Type ||
CSuite_Sel || inputString)

7. Packet Formats

This section defines the packet format of the EAP-GPSK messages.

7.1. Header Format

The EAP-GPSK header has the following structure:

```
--- bit offset --->
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |   Identifier  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    OP-Code    |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
...                         Payload                          ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
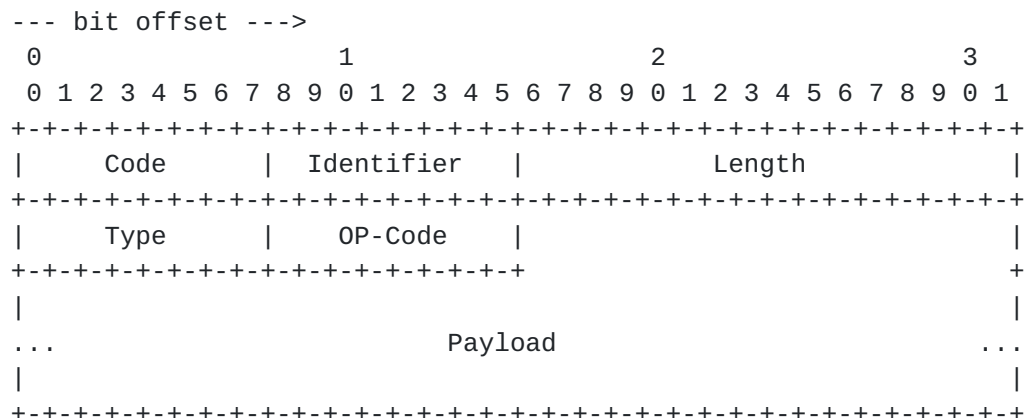
Figure 5

The Code, Identifier, Length, and Type fields are all part of the EAP

header, and defined in [RFC3748]. IANA has allocated EAP Method Type
XX for EAP-GPSK, thus the Type field in the EAP header MUST be XX.

The OP-Code field is one of four values:

o 0x01 : GPSK-1
o 0x02 : GPSK-2
o 0x03 : GPSK-3
o 0x04 : GPSK-4

7.2. Ciphersuite Formatting

Ciphersuites are encoded as 6-octet arrays. The first three octets
indicate the CSuite/Vendor field. For vendor-specific ciphersuites,
this represents the vendor OID. The last three octets indicate the
CSuite/Specifier field, which identifies the particular ciphersuite.
The 3-byte CSuite/Vendor value 0x000000 indicates ciphersuites
allocated by the IETF.

Graphically, they are represented as

```
--- bit offset --->
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       CSuite/Vendor = 0x000000 or OID        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      CSuite / Specifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
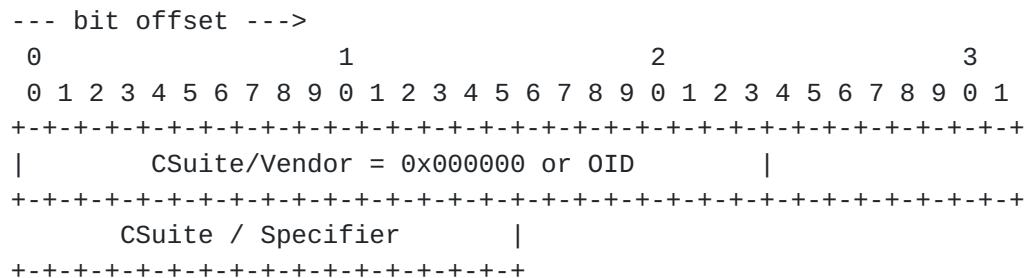
                            Figure 6

CSuite_Sel is encoded as a 6-octet ciphersuite CSuite/Vendor and
CSuite/Specifier pair.

CSuite_List is a variable-length octet array of ciphersuites. It is
encoded by concatenating encoded ciphersuite values. Its length in
octets MUST be a multiple of 6.

7.3. Payload Formatting

Payload formatting is based on the protocol exchange description in
Section 3.

The GPSK-1 payload format is defined as follows:

```
    --- bit offset --->
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |       length(ID_Server)       |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
    |                                                               |
    ...                            ID_Server                      ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ...                     32-octet RAND_Server                  ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      length(CSuite_List)      |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
    |                                                               |
    ...                          CSuite_List                      ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
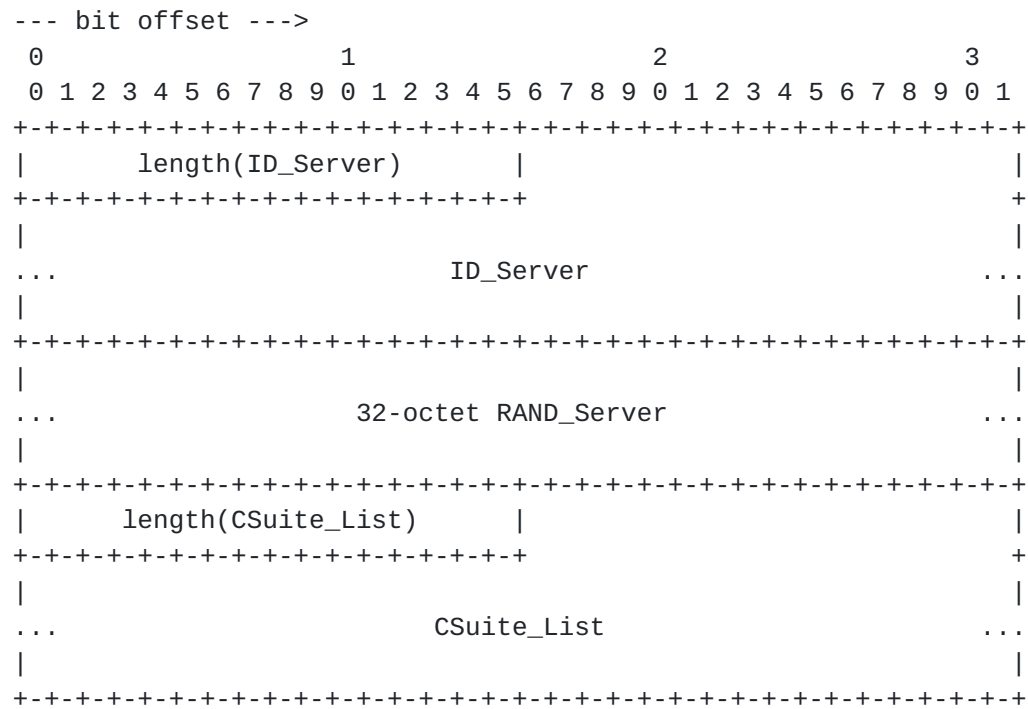
Figure 7: GPSK-1 Payload

The GPSK-2 payload format is defined as follows:

```
    --- bit offset --->
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |       length(ID_Client)       |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
    |                                                               |
    ...                          ID_Client                       ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |       length(ID_Server)       |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
    |                                                               |
    ...                          ID_Server                       ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ...                    32-octet RAND_Client                  ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ...                    32-octet RAND_Server                  ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      length(CSuite_List)      |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
    |                                                               |
    ...                         CSuite_List                      ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                           CSuite_Sel                          |
    +                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                               |     length(PD_Payload_1)      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ...                     optional PD_Payload_1                ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ...                     KS-octet payload MAC                 ...
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
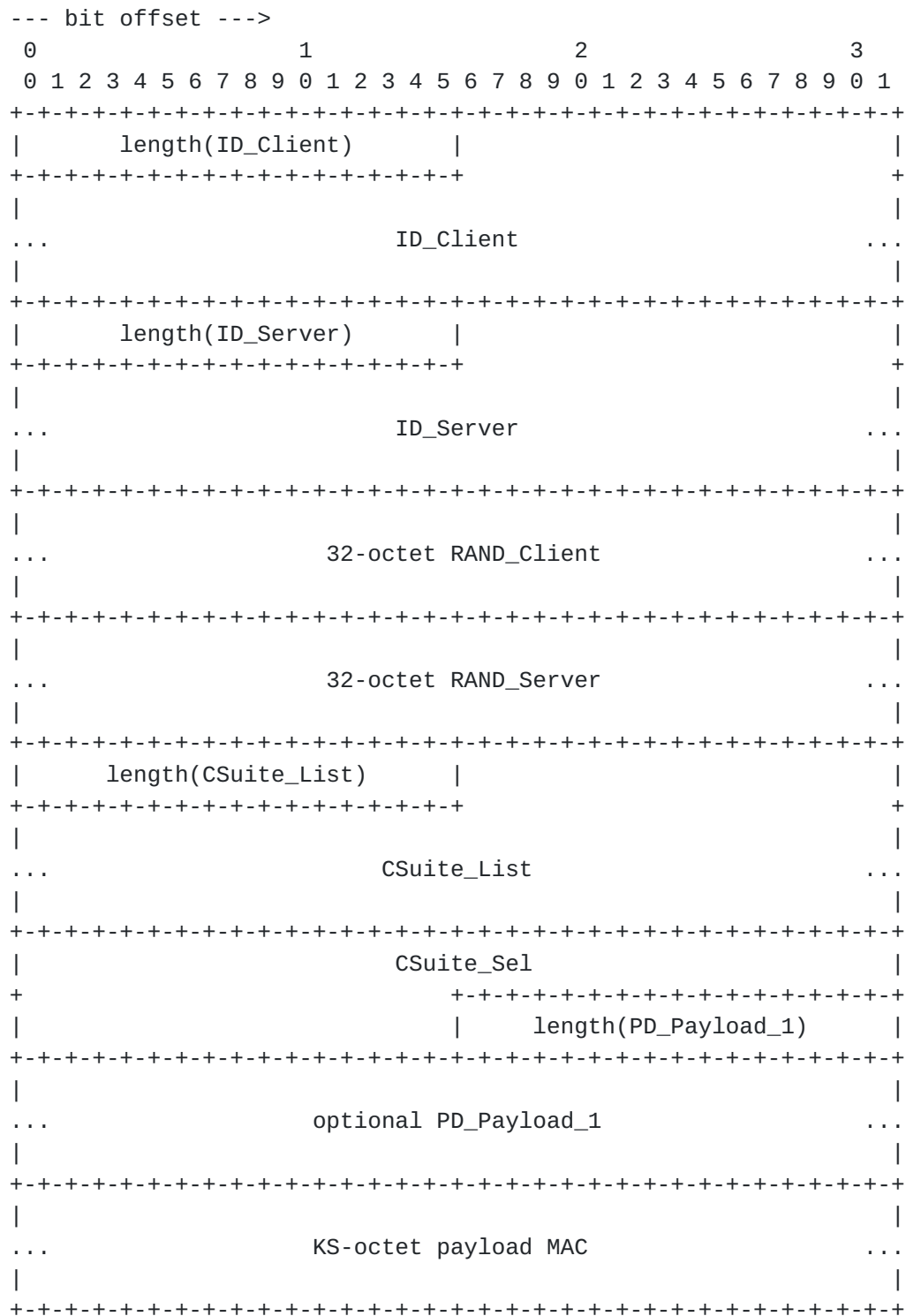
Figure 8: GPSK-2 Payload

If the optional protected data payload is not included, then
length(PD_Payload)=0 and the PD payload is excluded.
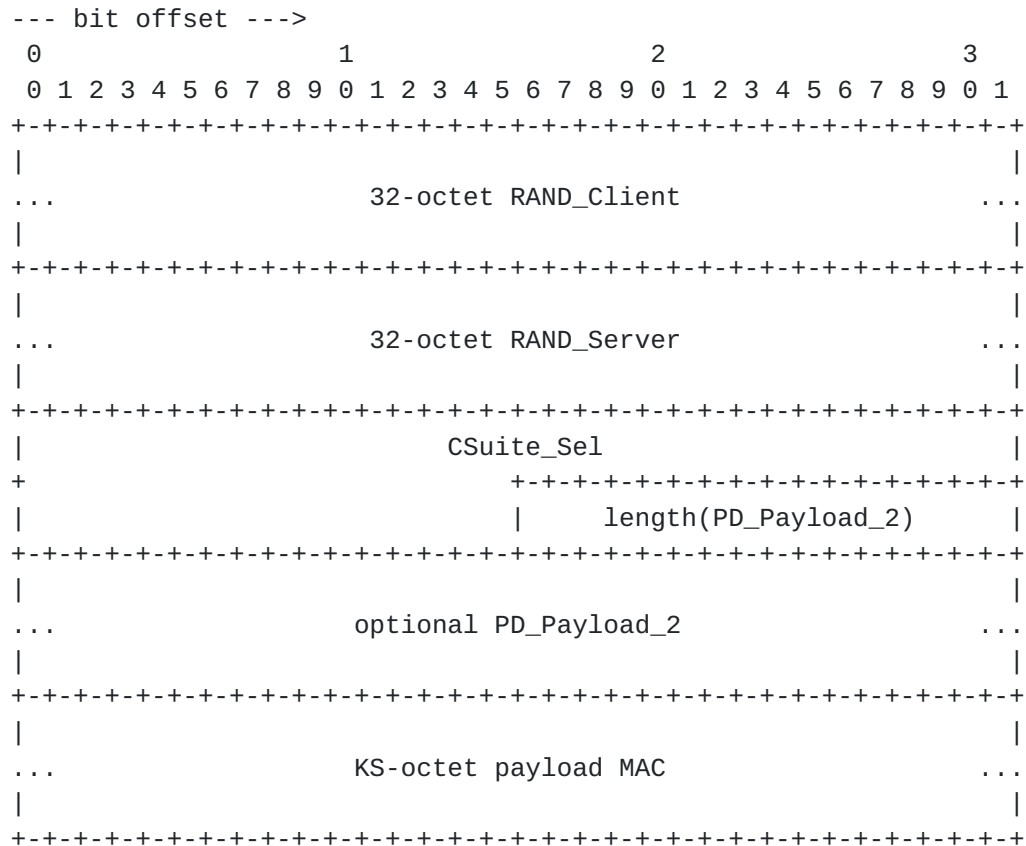
The GPSK-3 payload is defined as follows:

```
--- bit offset --->
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
...                    32-octet RAND_Client                   ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
...                    32-octet RAND_Server                   ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            CSuite_Sel                         |
+                           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           |      length(PD_Payload_2)         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
...                    optional PD_Payload_2                  ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
...                    KS-octet payload MAC                   ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
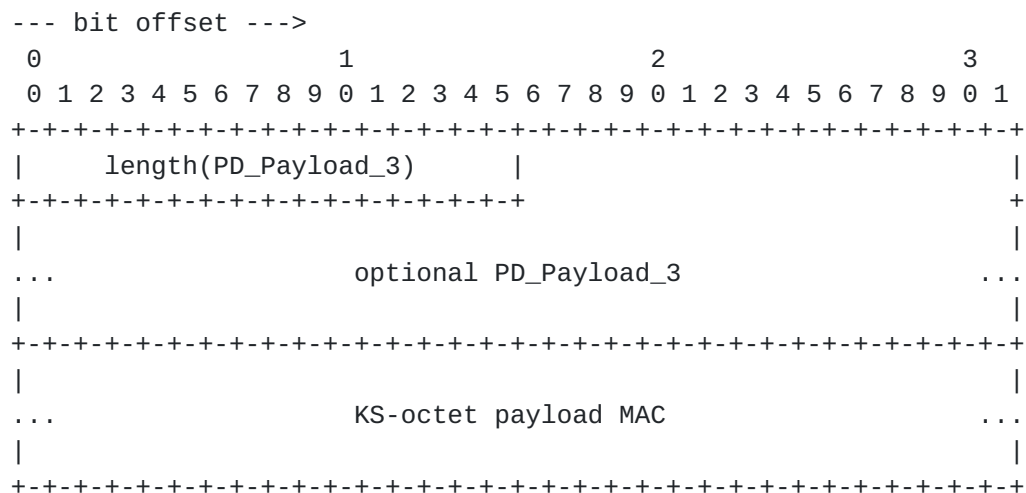
Figure 9: GPSK-3 Payload

If the optional protected data payload is not included, then
length(PD_Payload)=0 and the PD payload is excluded.

The GPSK-4 payload format is defined as follows:

```
--- bit offset --->
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     length(PD_Payload_3)       |                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                             +
|                                                             |
...                     optional PD_Payload_3                 ...
|                                                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                             |
...                     KS-octet payload MAC                  ...
|                                                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                    Figure 10: GPSK-4 Payload

   If the optional protected data payload is not included, then
   length(PD_Payload)=0 and the PD payload is excluded. The MAC MUST
   always be included, regardless of the presence of PD_Payload_3.

7.4. Protected Data

   The protected data blocks are a generic mechanism for the client and
   server to securely exchange data. If the specified ciphersuite has a
   NULL encryption primitive, then this channel only offers
   authenticity, and not confidentiality.

   These payloads are encoded as the concatenation of type-length-value
   (TLV) tripples.

   Type values are encoded as a 6-octet string and represented by a
   3-octet vendor and 3-octet specifier field. The vendor field
   indicates the type as either standards-specified or vendor-specific.
   If these three octets are 0x000000, then the value is standards-
   specified, and any other value represents a vendor-specific OID.

   The specifier field indicates the actual type. For vendor field
   0x000000, the specifier field is maintained by IANA. For any other
   vendor field, the specifier field is maintained by the vendor.

   Length fields are specified as 2-octet integers in network byte
   order, and reflect only the length of the value, and do not include
   the length of the type and length fields.

   Graphically, this can be depicted as follows:

```
--- bit offset --->
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       PData/Vendor               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        PData/Specifier         |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
...                      PD_Payload Value                    ...
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
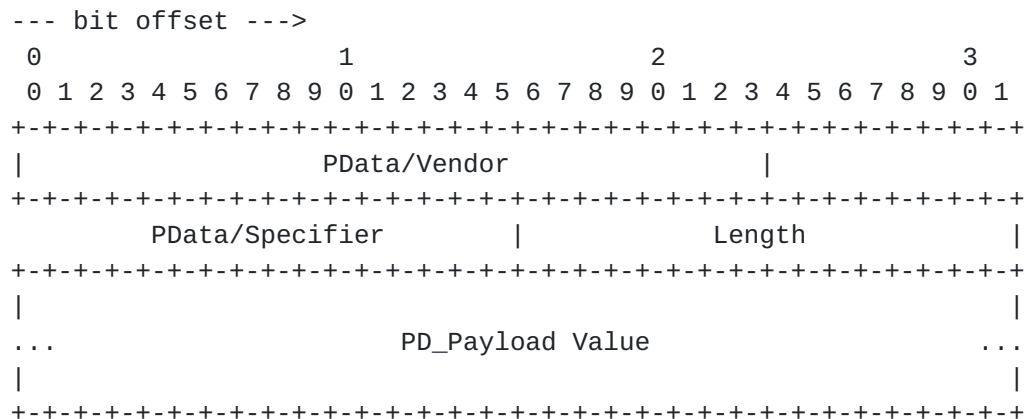
For PData/Vendor field 0x000000, the following PData/Specifier fields
are defined:
o   0x000000 : Reserved
o   0x000001 : Protected Results Indication
o   0x000002 through 0xFFFFFF : Unallocated

[Editor's Note: Text for protected results indication needs to be
added here.]

8. Security Considerations

[RFC3748] highlights several attacks that are possible against EAP
since EAP itself does not provide any security.

This section discusses the claimed security properties of EAP-GPSK as
well as vulnerabilities and security recommendations in the threat
model of [RFC3748].

8.1. Mutual Authentication

EAP-GPSK provides mutual authentication.

The server believes that the peer is authentic because it can
calculate a valid MAC and the peer believes that the server is
authentic because it can calculate another valid MAC.

The key used for mutual authentication is computed again based on the
long-term secret PSK that has to provide sufficient entropy and
therefore sufficient strength. In this way EAP-GPSK is no different
than other authentication protocols based on pre-shared keys.

8.2. Protected Result Indications

   EAP-GPSK offers the capability to exchange protected result
   indications using the protected data payloads.

8.3. Integrity Protection

   EAP-GPSK provides integrity protection based on the ciphersuites
   suggested in this document.

8.4. Replay Protection

   EAP-GPSK provides replay protection of its mutual authentication part
   thanks to the use of random numbers RAND_Server and RAND_P. Since
   RAND_Server is 128 bit long, one expects to have to record 2**64
   (i.e., approximately 1.84*10**19) EAP-GPSK successful authentication
   before an protocol run can be replayed. Hence, EAP-GPSK provides
   replay protection of its mutual authentication part as long as
   RAND_Server and RAND_Client are chosen at random, randomness is
   critical for security.

8.5. Reflection attacks

   EAP-GPSK provides protection against reflection attacks in case of an
   extended authentication because of the messages are constructed in a
   different fashion.

8.6. Dictionary Attacks

   EAP-GPSK relies on a long-term shared secret (PSK) that MUST be based
   on at least 128 bits of entropy to guarantee security against
   dictionary attacks. Users who use passwords or weak keys are not
   guaranteed security against dictionary attacks. Derivation of the
   long-term shared secret from a password is highly discouraged.

8.7. Key Derivation

   EAP-GPSK supports key derivation as shown in Section 4.

8.8. Denial of Service Resistance

   Denial of Service resistance (DoS) has not been a design goal for
   EAP-GPSK.

   It is however believed that EAP-GPSK does not provide any obvious and
   avoidable venue for such attacks.

   It is worth noting that the server has to maintain some state when it

engages in an EAP-GPSK conversation, namely to generate and to
remember the 16-byte RAND_S. This should however not lead to resource
exhaustion as this state and the associated computation are fairly
lightweight.

It is recommended that EAP-GPSK does not allow EAP notifications to
be interleaved in its dialog to prevent potential DoS attacks.
Indeed, since EAP Notifications are not integrity protected, they can
easily be spoofed by an attacker. Such an attacker could force a
peer that allows EAP Notifications to engage in a discussion which
would delay his authentication or result in the peer taking
unexpected actions (e.g., in case a notification is used to prompt
the peer to do some "bad" action).

It is up to the implementation of EAP-GPSK or to the peer and the
server to specify the maximum number of failed cryptographic checks
that are allowed.

8.9. Session Independence

Thanks to its key derivation mechanisms, EAP-GPSK provides session
independence: passive attacks (such as capture of the EAP
conversation) or active attacks (including compromise of the MSK or
EMSK) do not enable compromise of subsequent or prior MSKs or EMSKs.
The assumption that RAND_Client and RAND_Server are random is central
for the security of EAP-GPSK in general and session independance in
particular.

8.10. Exposition of the PSK

EAP-GPSK does not provide perfect forward secrecy. Compromise of the
PSK leads to compromise of recorded past sessions.

Compromise of the PSK enables the attacker to impersonate the peer
and the server and it allows the adversary to compromise future
sessions.

EAP-GPSK provides no protection against a legitimate peer sharing its
PSK with a third party. Such protection may be provided by
appropriate repositories for the PSK, which choice is outside the
scope of this document. The PSK used by EAP-GPSK must only be shared
between two parties: the peer and the server. In particular, this
PSK must not be shared by a group of peers communicating with the
same server.

The PSK used by EAP-GPSK must be cryptographically separated from
keys used by other protocols, otherwise the security of EAP-GPSK may
be compromised.

8.11. Fragmentation

   EAP-GPSK does not support fragmentation and reassembly since the
   message size is kept small.

8.12. Channel Binding

   This document enables the ability to exchange channel binding
   information. It does not, however, define the encoding of channel
   binding information in the document.

8.13. Fast Reconnect

   EAP-GPSK does not provide the fast reconnect capability since this
   method is already at the lower limit of the number of roundtrips and
   the cryptographic operations.

8.14. Identity Protection

   Identity protection is not specified in this document. Extensions
   can be defined that enhanced this protocol to provide this feature.

8.15. Protected Ciphersuite Negotiation

   EAP-GPSK provides protected ciphersuite negotiation via the
   indication of available ciphersuites by the server in the first
   message and a confirmation by the client in the subsequent message.

8.16. Confidentiality

   Although EAP-GPSK provides confidentiality in its protected data
   payloads, it cannot claim to do so as per Section 7.2.1 of [RFC3748].

8.17. Cryptographic Binding

   Since EAP-GPSK does not tunnel another EAP method, it does not
   implement cryptographic binding.

9. IANA Considerations

   This document requires IANA to allocate a new EAP Type for EAP-GPSK.

   This document requires IANA to create a new registry for ciphersuites
   and protected data types. IANA is furthermore instructed to add the
   specified ciphersuites and protected data types to this registry as
   defined in this document. Values can be added or modified with
   informational RFCs defining either block-based or hash-based

ciphersuites. Each ciphersuite needs to provide processing rules and
needs to specify how the following algorithms are instantiated:
Encryption, Integrity and Key Derivation. Additionally, the
preferred key size needs to be specified.

The following layout represents the initial ciphersuite CSuite/
Specifier registry setup:

o 0x000000 : Reserved
o 0x000001 : AES-CBC-128, AES-CMAC-128, GKDF-128
o 0x000002 : NULL, HMAC-SHA256, GKDF-256
o 0x000003 through 0xFFFFFF : Unallocated

The following is the initial protected data PData/Specifier registry
setup:

o 0x000000 : Reserved
o 0x000001 : Protected Results Indication
o 0x000002 through 0xFFFFFF : Unallocated

10. Contributors

This work is a joint effort of the EAP Method Update (EMU) design
team of the EMU Working Group that was created to develop a mechanism
based on strong shared secrets that meets RFC 3748 [RFC3748] and RFC
4017 [RFC4017] requirements. The contributors (in alphabetical
order) include:

o Jari Arkko
o Mohamad Badra
o Uri Blumenthal
o Charles Clancy
o Lakshminath Dondeti
o David McGrew
o Joe Salowey
o Sharma Suman
o Hannes Tschofenig
o Jesse Walker

Finally, we would like to thank Thomas Otto for his draft reviews,
feedback and text contributions.

11. Acknowledgment

We would like to thank

o Jouni Malinen and Bernard Aboba for their early draft comments in
   June 2006.
o Lakshminath Dondeti, David McGrew, Bernard Aboba, Michaela
   Vanderveen and Ray Bell for their input to the ciphersuite
   discussions between July and August 2006.
o Lakshminath for his detailed draft review (sent to the EMU ML on
   the 12th July 2006).

12. Open Issues

The list of open issues can be found at:
http://www.tschofenig.com:8080/eap-gpsk/

A first prototype implementation by Jouni Malinen can be found at:
http://hostap.epitest.fi/releases/snapshots/

13. References

13.1. Normative References

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

[RFC2486bis]
           Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The
           Network Access Identifier",
           draft-ietf-radext-rfc2486bis-06 (work in progress),
           July 2005.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1
           (SHA1)", RFC 3174, September 2001.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H.
           Levkowetz, "Extensible Authentication Protocol (EAP)",
           RFC 3748, June 2004.

13.2. Informative References

[I-D.clancy-eap-pax]
           Clancy, C. and W. Arbaugh, "EAP Password Authenticated
           Exchange", draft-clancy-eap-pax-11 (work in progress),
           September 2006.

[I-D.bersani-eap-psk]
           Tschofenig, H. and F. Bersani, "The EAP-PSK Protocol: a
           Pre-Shared Key EAP Method", draft-bersani-eap-psk-11 (work

                    in progress), June 2006.

   [I-D.otto-emu-eap-tls-psk]
                    Otto, T. and H. Tschofenig, "The EAP-TLS-PSK
                    Authentication Protocol", draft-otto-emu-eap-tls-psk-01
                    (work in progress), October 2006.

   [I-D.vanderveen-eap-sake]
                    Vanderveen, M. and H. Soliman, "Extensible Authentication
                    Protocol Method for Shared-secret Authentication and Key
                    Establishment (EAP-SAKE)", draft-vanderveen-eap-sake-02
                    (work in progress), May 2006.

   [I-D.ietf-eap-keying]
                    Aboba, B., "Extensible Authentication Protocol (EAP) Key
                    Management Framework", draft-ietf-eap-keying-15 (work in
                    progress), October 2006.

   [RFC4017]   Stanley, D., Walker, J., and B. Aboba, "Extensible
                    Authentication Protocol (EAP) Method Requirements for
                    Wireless LANs", RFC 4017, March 2005.

   [CMAC]   National Institute of Standards and Technology,
                    "Recommendation for Block Cipher Modes of Operation: The
                    CMAC Mode for Authentication", Special Publication
                    (SP) 800-38B, May 2005.

   [CBC]   National Institute of Standards and Technology,
                    "Recommendation for Block Cipher Modes of Encryption.
                    Methods and Techniques.", Special Publication (SP) 800-
                    38A, December 2001.

Authors' Addresses

   T. Charles Clancy
   DoD Laboratory for Telecommunication Sciences
   8080 Greenmeade Drive
   College Park, MD 20740
   USA

   Email: clancy@ltsnet.net

      Hannes Tschofenig
      Siemens
      Otto-Hahn-Ring 6
      Munich, Bavaria 81739
      Germany

      Email: Hannes.Tschofenig@siemens.com
      URI: http://www.tschofenig.com

Full Copyright Statement

Intellectual Property

Acknowledgment