                     The EAP-PSK Protocol:
    A Pre-Shared Key Extensible Authentication Protocol (EAP) Method

Status of This Memo

Copyright Notice

IESG Note

   This RFC is not a candidate for any level of Internet Standard. The
   IETF disclaims any knowledge of the fitness of this RFC for any
   purpose and in particular notes that the decision to publish is not
   based on IETF review for such things as security, congestion control,
   or inappropriate interaction with deployed protocols. The RFC Editor
   has chosen to publish this document at its discretion. Readers of
   this document should exercise caution in evaluating its value for
   implementation and deployment. See RFC 3932 for more information.

   The IESG thinks that this work is related to IETF work done in WGs
   EMU and EAP, but this does not prevent publishing.

Abstract

   This document specifies EAP-PSK, an Extensible Authentication
   Protocol (EAP) method for mutual authentication and session key
   derivation using a Pre-Shared Key (PSK). EAP-PSK provides a
   protected communication channel when mutual authentication is
   successful for both parties to communicate over. This document
   describes the use of this channel only for protected exchange of
   result indications, but future EAP-PSK extensions may use the channel
   for other purposes. EAP-PSK is designed for authentication over
   insecure networks such as IEEE 802.11.

Table of Contents

1. Introduction

1.1. Design Goals for EAP-PSK

   The Extensible Authentication Protocol (EAP) [3] provides an
   authentication framework that supports multiple authentication
   methods.

   This document specifies an EAP method, called EAP-PSK, that uses a
   Pre-Shared Key (PSK).

   EAP-PSK was developed at France Telecom R&D in 2003-2004. It is
   published as an RFC for the general information of the Internet
   community and to allow independent implementations.

   Because PSKs are of frequent use in security protocols, other
   protocols may also refer to a PSK or contain this word in their name.
   For instance, Wi-Fi Protected Access (WPA) [48] specifies an
   authentication mode called "WPA-PSK". EAP-PSK is distinct from these
   protocols and should not be confused with them.

   Design goals for EAP-PSK were:

   o Simplicity: EAP-PSK should be easy to implement and deploy without
      any pre-existing infrastructure. It should be available quickly
      because recently-released protocols, such as IEEE 802.11i [27],
      employ EAP in a different threat model than PPP [44] and thus
      require "modern" EAP methods.

   o Wide applicability: EAP-PSK should be suitable to authenticate
      over any network, and in particular over IEEE 802.11 [28] wireless
      LANs.

   o Security: EAP-PSK should be conservative in its cryptographic
      design.

   o Extensibility: EAP-PSK should be easily extensible.

1.1.1. Simplicity

   For the sake of simplicity, EAP-PSK relies on a single cryptographic
   primitive, AES-128 [7].

   Restriction to such a primitive, and in particular, not using
   asymmetric cryptography like Diffie-Hellman key exchange, makes EAP-
   PSK:

   o Easy to understand and implement while avoiding cryptographic
      negotiations.

   o Lightweight and well suited for any type of device, especially
      those with little processing power and memory.

   However, as further discussed in Section 8, this prevents EAP-PSK
   from offering advanced features such as identity protection, password
   support, or Perfect Forward Secrecy (PFS). This choice has been
   deliberately made as a trade-off between simplicity and security.

   For the sake of simplicity, EAP-PSK has also chosen a fixed message
   format and not a Type-Length-Value (TLV) design.

1.1.2. Wide Applicability

   EAP-PSK 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 [3].

1.1.3. Security

   Since the design of authenticated key exchange is notoriously known
   to be hard and error prone, EAP-PSK tries to avoid inventing any new
   cryptographic mechanism. It attempts instead to build on existing
   primitives and protocols that have been reviewed by the cryptographic
   community.

1.1.4. Extensibility

   EAP-PSK explicitly provides a mechanism to allow future extensions
   within its protected channel (see Section 3.3). Thanks to this
   mechanism, EAP-PSK will be able to provide more sophisticated
   services as the need to do so arises.

1.2. Terminology

   Authentication, Authorization, and Accounting (AAA)
           Please refer to [10] for more details.

   AES-128   A block cipher specified in the Advanced Encryption
             Standard [7].

   Authentication Key (AK)
           A 16-byte key derived from the PSK that the EAP peer and
           server use to mutually authenticate.

AKEP2   An authenticated key exchange protocol; please refer to
           [14] for more details.

Backend Authentication Server
           An entity that provides an authentication service to an
           Authenticator. When used, this server typically executes
           EAP methods for the Authenticator. (This terminology is
           also used in [26], and has the same meaning in this
           document.)

CMAC   Cipher-based Message Authentication Code. It is the
           authentication mode of operation of AES recommended by NIST
           in [8].

Extensible Authentication Protocol (EAP)
           Defined in [3].

EAP Authenticator (or simply Authenticator)
           The end of the EAP link initiating the EAP authentication
           methods. (This terminology is also used in [26], and has
           the same meaning in this document.)

EAP peer (or simply peer)
           The end of the EAP link that responds to the Authenticator.
           (In [26], this end is known as the Supplicant.)

EAP server (or simply server)
           The entity that terminates the EAP authentication with the
           peer. When there is no Backend Authentication Server, this
           term refers to the EAP Authenticator. Where the EAP
           Authenticator operates in pass-through mode, it refers to
           the Backend Authentication Server.

EAX   An authenticated-encryption with associated data mode of
           operation for block ciphers [4].

Extended Master Session Key (EMSK)
           Additional keying material derived between the EAP peer and
           server that is exported by the EAP method. The EMSK is
           reserved for future uses that are not defined yet and is
           not provided to a third party. Please refer to [9] for
           more details.
           EAP-PSK generates a 64-byte EMSK.

Initialization Vector (IV)
           A quantity of at least 64 bytes, suitable for use in an
           initialization vector field, that is derived between the
           peer and EAP server. Since the IV is a known value in

methods such as EAP-TLS [11], it cannot be used by itself
for computation of any quantity that needs to remain
secret. As a result, its use has been deprecated and EAP
methods are not required to generate it. Please refer to
[9] for more details.
EAP-PSK does not generate an IV.

Key-Derivation Key (KDK)
        A 16-byte key derived from the PSK that the EAP peer and
        server use to derive session keys (namely, the TEK, MSK,
        and EMSK).

Message Authentication Code (MAC)
        Informally, the purpose of a MAC is to provide assurances
        regarding both the source of a message and its integrity
        [40].   IEEE 802.11i uses the acronym MIC (Message Integrity
        Check) to avoid confusion with the other meaning of the
        acronym MAC (Medium Access Control).

Master Session Key (MSK)
        Keying material that is derived between the EAP peer and
        server and exported by the EAP method. In existing
        implementations, a AAA server acting as an EAP server
        transports the MSK to the Authenticator [9].
        EAP-PSK generates a 64-byte MSK.

Network Access Identifier (NAI)
        Identifier used to identify the communicating parties [2].

One Key CBC-MAC 1 (OMAC1)
        A method to generate a Message Authentication Code [29].
        CMAC is the name under which NIST has standardized OMAC1.

Perfect Forward Secrecy (PFS)
        The confidence that the compromise of a long-term private
        key does not compromise any earlier session keys. In other
        words, once an EAP dialog is finished and its corresponding
        keys are forgotten, even someone who has recorded all of
        the data from the connection and gets access to all of the
        long-term keys of the peer and the server cannot
        reconstruct the keys used to protect the conversation
        without doing a brute-force search of the session key
        space.

        EAP-PSK does not have this property.

   Pre-Shared Key (PSK)
            A Pre-Shared Key simply means a key in symmetric
            cryptography. This key is derived by some prior mechanism
            and shared between the parties before the protocol using it
            takes place. It is merely a bit sequence of given length,
            each bit of which has been chosen at random uniformly and
            independently. For EAP-PSK, the PSK is the long-term 16-
            byte credential shared by the EAP peer and server.

   Protected Result Indication
            Please refer to Section 7.16 of [3] for a definition of
            this term. This feature has been introduced because EAP-
            Success/Failure packets are unidirectional and are not
            protected.

   Transient EAP Key (TEK)
            A session key that is used to establish a protected channel
            between the EAP peer and server during the EAP
            authentication exchange. The TEK is appropriate for use
            with the ciphersuite negotiated between the EAP peer and
            server to protect the EAP conversation. Note that the
            ciphersuite used to set up the protected channel between
            the EAP peer and server during EAP authentication is
            unrelated to the ciphersuite used to subsequently protect
            data sent between the EAP peer and Authenticator [9].
            EAP-PSK uses a 16-byte TEK for its protected channel, which
            is the only ciphersuite available between the EAP peer and
            server to protect the EAP conversation. This ciphersuite
            uses AES-128 in the EAX mode of operation.

1.3. Conventions

   All numbers presented in this document are considered in network-byte
   order.

   || denotes concatenation of strings (and not the logical OR).

   MAC(K, String) denotes the MAC of String under the key K (the
   algorithm used in this document to compute the MACs is CMAC with AES-
   128; see Section 3.2).

   [String] denotes the concatenation of String with the MAC of String
   calculated as specified by the context. Hence, we have, with K
   specified by the context: [String]=String||MAC(K,String)

   ** denotes integer exponentiation.

"i" denotes the unsigned binary representation on 16 bytes of the
integer i in network byte order. Therefore, this notation only makes
sense when i is between 0 and 2**128-1.

<i> denotes the unsigned binary representation on 4 bytes of the
integer i in network byte order. Therefore, this notation only makes
sense when i is between 0 and 2**32-1.

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 [1].

1.4. Related Work

   At the time this document is written, only three EAP methods are
   standards track EAP methods per IETF terminology (see [17]), namely:

   o MD5-Challenge (EAP-Request/Response type 4), defined in [3], which
      uses a MD5 challenge similar to [45].

   o OTP (EAP-Request/Response type 5), defined in [3], which aims at
      providing One-Time Password support similar to [22] and [39].

   o GTC (EAP-Request/Response type 6), defined in [3], which aims at
      providing Generic Token Card Support.

   Unfortunately, all three methods are deprecated for security reasons
   that are explained in part in [3].

   Myriads of EAP methods have, however, been otherwise proposed:

   o One as an experimental RFC (EAP-TLS [11]), which therefore is not
      a standard (see [25]).

   o Some as individual Internet-Draft submissions (e.g., [42] or this
      document).

   o And some even undocumented (e.g., Rob EAP, which has EAP-Request/
      Response type 31).

   However, no secure and mature Pre-Shared Key EAP method is yet easily
   and widely available, which is all the more regrettable because Pre-
   Shared Key methods are the most basic ones!

   The existing proposals for a future Pre-Shared Key EAP method are
   briefly reviewed hereafter (please refer to [16] for a more thorough
   synthesis of EAP methods).

Among these proposals, there are some that:

o Are broken from a security point of view, e.g.:

   * LEAP, which is specified in [38] and whose vulnerabilities are
      discussed in [49].

   * EAP-MSCHAPv2, which is specified in [34] and whose
      vulnerabilities are indirectly discussed in [43].

o Essentially require additional infrastructure, e.g., EAP-SIM [24],
   EAP-AKA [12], or OTP/token card methods like [31].

o Are not shared key methods but are often confused with them,
   namely, the password methods, e.g., EAP-SRP [18] or SPEKE [30],
   whose wide adoption very unfortunately seems to be hindered by
   Intellectual Property Rights issues.

o Are generic tunneling methods, which do not essentially rely on
   Pre-Shared Keys as they require a public-key certificate for the
   server and allow the peer to authenticate with whatever EAP method
   or even other non-EAP authentication mechanisms, namely, [32] and
   [21].

o Are abandoned but have provided the basis for EAP-PSK, namely,
   EAP-Archie [47].

o Are possible alternatives to EAP-PSK (i.e., claimed to be secure
   and subject of active work):

   * EAP-FAST [42].

   * EAP-IKEv2 [46].

   * EAP-TLS (when shared key/password support is added to TLS; see
      [50]).

EAP-PSK differs from the aforementioned methods on the following
points:

o No attacks on EAP-PSK within its threat model have yet been found.

o EAP-PSK was not designed to leverage a pre-existing
   infrastructure. Thus, it does not inherit potential limitations
   of such an infrastructure and it should be easier to deploy "from
   scratch".

o EAP-PSK wished to avoid IPR blockages.

   o EAP-PSK does not have any dependencies on protocols other than
      EAP.

   o EAP-PSK was restricted to simply proposing a Pre-Shared Key method
      with symmetric cryptography

      * To remain simple to understand and implement

      * To avoid potentially complex configurations and negotiations

   o EAP-PSK was designed with efficiency in mind.

2. Protocol Overview

    Figure 1 presents an overview of the EAP-PSK key hierarchy.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++ ---+
|                                                          |   ^
|           EAP-PSK Protocol: a Pre-Shared Key EAP Method  |   |
|                                                          |   |
|                    +----------+                          |   |
|                    |   PSK    |                          |   |
|                    |(16 bytes)|                          |   |
|                    +----------+                          |   |
|                         |                                |   |
|                         v                                |   |
|           ************************                       |   |
|           *Modified Counter Mode*                        |   |
|           ************************                       |   |
|                |       |                                 |   |
|                v       v                                 |   |
|           +---------+ +---------+ +----------------+      |   |
|           |   AK    | |   KDK   | |     RAND_P     |      |   |
|           |(16 bytes)| |(16 bytes)| |   (16 bytes)   |     |   |
|           +---------+ +---------+ +----------------+      |   |
|                |           |             |               |   |
|                |    +-----------+        |               |   |
|          +--------+ |Plain Text |        |               |   |
|  +-------+|Header H| |Var.Length |       |               |   |
|  |Nonce N||22 bytes|+-----------+       v               v   Local
|  |4 bytes|+-------+      |    ***********************   to EAP
|  +-------+   |  +--------+  +------*Modified Counter Mode* Method
|      |       v v        v   ***********************       |
|      |    *******   +--------+ |64            |64         |
|      |    * EAX *-------|TEK    | |bytes         |bytes      |
|      +-->*******   |16 bytes| |              |           |
|           |        +--------+ |              |           |
|      +-----+----+            |              |           |
|      v          v            |              |           |
| +--------+ +------------------+           |              |           |
| |Tag     | |Cipher Text Payload|          |              |           |
| |16 bytes| |Variable length L  |          |              |           |
| +--------+ +------------------+           |              |           V
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++ ---+
                     |              |                           ^
                +-+-+-+-+-++  +-+-+-+-+-++                      |
                |MSK       |  |EMSK     |                       |
                |          |  |         |                Exported|
                +-+-+-+-+-++  +-+-+-+-+-++               by EAP |
```

```
                   |                   |              Method |
                   V                   V                     |
         *************************                     V
         *   AAA   Key Derivation *              ---+
         *   Naming & Binding     *
         *************************
```

                Figure 1: EAP-PSK Key Hierarchy Overview

2.1. EAP-PSK Key Hierarchy

   This section presents the key hierarchy used by EAP-PSK. This
   hierarchy is inspired by the EAP key hierarchy described in [9].

2.1.1. The PSK

   The PSK is shared between the EAP peer and the EAP server.

   EAP-PSK assumes that the PSK is known only to the EAP peer and EAP
   server. The security properties of the protocol are compromised if
   it has wider distribution. Please note that EAP-PSK shares this
   property with all other symmetric key methods (including all
   password-based methods).

   EAP-PSK also assumes the EAP server and EAP peer identify the correct
   PSK to use with each other thanks to their respective NAIs. This
   means that there MUST only be at most one PSK shared between an EAP
   server using a given server NAI and an EAP peer using a given peer
   NAI.

   This PSK is used, as shown in Figure 2, to derive two 16-byte static
   long-lived subkeys, respectively called the Authentication Key (AK)
   and the Key-Derivation Key (KDK). This derivation should only be
   done once: it is called the key setup. See Section 3.1 for an
   explanation of why PSK is not used as a static long-lived key, but
   only as the initial keying material for deriving the static long-
   lived keys, AK and KDK, which are actually used by the protocol EAP-
   PSK.

```
            +--------------------------+
            |            PSK           |
            |        (16 bytes)        |
            +--------------------------+
                 |                  |
                 v                  v
  +--------------------------+  +--------------------------+
  |            AK            |  |            KDK           |
  |        (16 bytes)        |  |        (16 bytes)        |
  +--------------------------+  +--------------------------+
```
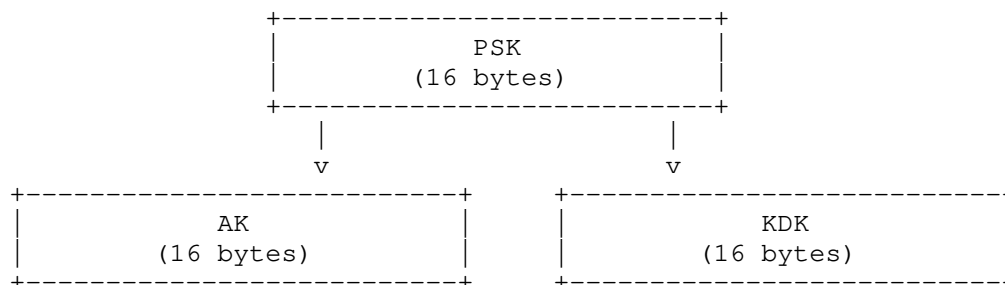
              Figure 2: Derivation of AK and KDK from the PSK

2.1.2. AK

   EAP-PSK uses AK to mutually authenticate the EAP peer and the EAP
   server.

   AK is a static long-lived key derived from the PSK; see Section 3.1.
   AK is not a session key.

   The EAP server and EAP peer identify the correct AK to use with each
   other thanks to their respective NAIs. This means that there MUST
   only be at most one AK shared between an EAP server using a given
   server NAI and an EAP peer using a given peer NAI. This is the case
   when there is at most one PSK shared between an EAP server using a
   given server NAI and an EAP peer using a given peer NAI; see
   Section 2.1.1.

   The EAP peer chooses the AK to use based on the EAP server NAI that
   has been sent by the EAP server in the first EAP-PSK message (namely,
   ID_S; see Section 4.1) and the EAP peer NAI it chooses to include in
   the second EAP-PSK message (namely, ID_P; see Section 4.1).

2.1.3. KDK

   EAP-PSK uses KDK to derive session keys shared by the EAP peer and
   the EAP server (namely, the TEK, MSK, and EMSK).

   KDK is a static long-lived key derived from the PSK; see Section 3.1.
   KDK is not a session key.

   The EAP server and EAP peer identify the correct AK to use with each
   other thanks to their respective NAIs. This means that there MUST
   only be at most one AK shared between an EAP server using a given
   server NAI and an EAP peer using a given peer NAI. This is the case

   when there is at most one PSK shared between an EAP server using a
   given server NAI and an EAP peer using a given peer NAI; see
   Section 2.1.1.

   The EAP peer chooses the AK to use based on the EAP server NAI that
   has been sent by the EAP server in the first EAP-PSK message (namely,
   ID_S; see Section 4.1) and the EAP peer NAI it chooses to include in
   the second EAP-PSK message (namely, ID_P; see Section 4.1).

2.2. The TEK

   EAP-PSK derives a 16-byte TEK thanks to a random number exchanged
   during authentication (RAND_P; see Section 5.1) and KDK.

   This TEK is used to implement a protected channel for both mutually
   authenticated parties to communicate over securely.

2.3. The MSK

   EAP-PSK derives a MSK thanks to a random number exchanged during
   authentication (RAND_P; see Section 5.1) and the KDK.

   The MSK is 64 bytes long, which complies with [3].

2.4. The EMSK

   EAP-PSK derives an EMSK thanks to a random number exchanged during
   authentication (RAND_P; see Section 5.1) and the KDK.

   The EMSK is 64 bytes long, which complies with [3].

2.5. The IV

   EAP-PSK does not derive any IV, which complies with [9].

3. Cryptographic Design of EAP-PSK

   EAP-PSK relies on a single cryptographic primitive, a block cipher,
   which is instantiated with AES-128. AES-128 takes a 16-byte Pre-
   Shared Key and a 16-byte Plain Text block as inputs. It outputs a
   16-byte Cipher Text block. For a detailed description of AES-128,
   please refer to [7].

   AES-128 has been chosen because:

   o It is standardized and implementations are widely available.

   o It has been carefully reviewed by the cryptographic community and
      is believed to be secure.

   Other block ciphers could easily be proposed for EAP-PSK, as EAP-PSK
   does not intrinsically depend on AES-128. The only parameters of
   AES-128 that EAP-PSK depends on are the AES-128 block and key size
   (16 bytes). For the sake of simplicity, EAP-PSK has, however, been
   chosen to restrict to a single mandatory block cipher and not allow
   the negotiation of other block ciphers. In the case that AES-128 is
   deprecated for security reasons, EAP-PSK should also be deprecated
   and a cut-and-paste EAP-PSK' should be defined with another block
   cipher. This EAP-PSK' should not be backward compatible with EAP-PSK
   because of the security issues with AES-128. EAP-PSK' should
   therefore use a different EAP-Request/Response Type number. With the
   EAP-Request/Response Type number space structure defined in [3], this
   should not be a problem. The use of a different EAP-Request/Response
   Type number for EAP-PSK' will prevent this new method from being
   vulnerable to chosen protocol attacks.

   EAP-PSK uses three cryptographic parts:

   o A key setup to derive AK and KDK from the PSK.

   o An authenticated key exchange protocol to mutually authenticate
      the communicating parties and derive session keys.

   o A protected channel protocol for both mutually authenticated
      parties to communicate over.

   Each part is discussed in more detail in the subsequent paragraphs.

3.1. The Key Setup

   EAP-PSK needs two cryptographically separated 16-byte subkeys for
   mutual authentication and session key derivation. Indeed, it is a
   rule of thumb in cryptography to use different keys for different
   applications.

   It could have implemented these two subkeys either by specifying a
   32-byte PSK that would then be split in two 16-byte subkeys, or by
   specifying a 16-byte PSK that would then be cryptographically
   expanded to two 16-byte subkeys.

   Because provisioning a 32-byte long-term credential is more
   cumbersome than a 16-byte one, and the strength of the derived
   session keys is 16 bytes either way, the latter option was chosen.

Hence, the PSK is only used by EAP-PSK to derive AK and KDK. This
derivation should be done only once, immediately after the PSK has
been provisioned. As soon as AK and KDK have been derived, the PSK
should be deleted. If the PSK is deleted, it should be done so
securely (see, for instance, [19] for guidance on secure deletion of
the PSK).

Derivation of AK and KDK from the PSK is called the key setup:

o   The input to the key setup is the PSK.

o   The outputs of the key setup are AK and KDK.

AK and KDK are derived from the PSK using the modified counter mode
of operation of AES-128. The modified counter mode is a length
increasing function, i.e., it expands one AES-128 input block into a
longer t-block output, where t>=2. This mode was chosen for the key
setup because it had already been chosen for the derivation of the
session keys (see Section 3.2).

The details of the derivation of AK and KDK from the PSK are shown in
Figure 3.

```
          +------------------------+
          |          "0"           |
          |  Input Block (16 bytes)|
          +------------------------+
                      |
                      v
              +---------------+
              |               |
              | AES-128(PSK,.)|
              |               |
              +---------------+
                      |
                      |
                  +------------------------------+
                  |                              |
                  v                              v
     +--------+  +---+            +--------+  +---+
     |c1="1"  |->|XOR|            |c2="2"  |->|XOR|
     |16 bytes|  +---+            |16 bytes|  +---+
     +--------+    |              +--------+    |
                   |                            |
              +---------------+            +---------------+
              |               |            |               |
              | AES-128(PSK,.)|            | AES-128(PSK,.)|
              |               |            |               |
              +---------------+            +---------------+
                      |                            |
                      |                            |
                      v                            v
     +----------------------+       +----------------------+
     |         AK           |       |         KDK          |
     |      (16 bytes)      |       |      (16 bytes)      |
     +----------------------+       +----------------------+
```
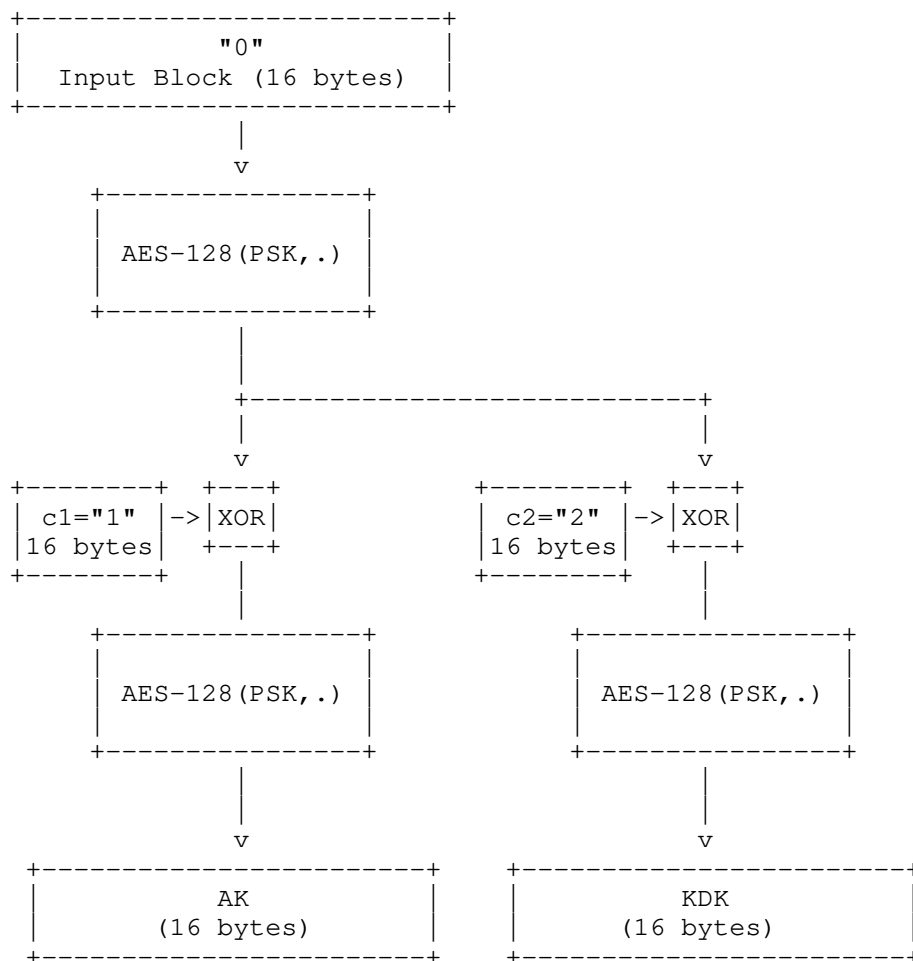
Figure 3: Derivation of AK and KDK from the PSK in Details

The input block is "0". For the sake of simplicity, this input block
has been chosen constant: it could have been set to a value depending
on the peer and the server (for instance, the XOR of their respective
NAIs appropriately truncated or zero-padded), but this did not seem
to add much security to the scheme, whereas it added complexity. Any
16-byte constant could have been chosen, as the security is not
supposed to depend on the particular value taken by the constant. "0"
was arbitrarily chosen.

3.2. The Authenticated Key Exchange

   The authentication protocol used by EAP-PSK is inspired by AKEP2,
   which is described in [14].

   AKEP2 consists of a one-and-a-half round-trip exchange, as shown in
   Figure 4, which is inspired by Figure 5 of [14].

```
Bob                                                           Alice
 |                             RA                                |
 |<-------------------------------------------------------------|
 |                                                               |
 |                                                               |
 |                       [B||A||RA||RB]                          |
 |------------------------------------------------------------->|
 |                                                               |
 |                                                               |
 |                          [A||RB]                              |
 |<-------------------------------------------------------------|
```
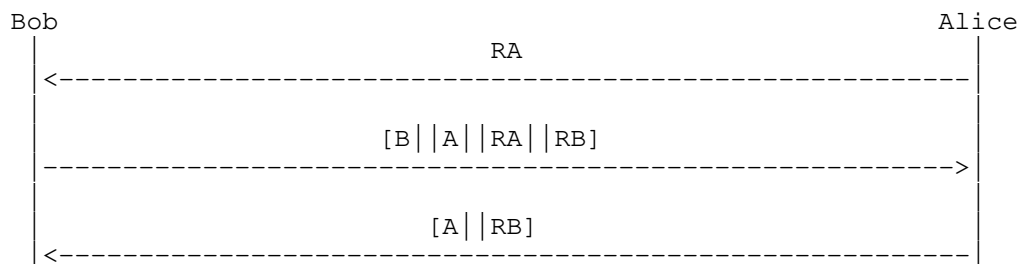
                     Figure 4: Overview of AKEP2

   It is also worth noting that [14] focuses on cryptography and not on
   designing a real-life protocol. Thus, as noted in subsection "Out-
   Of-Band-Data" of [14], Alice has to send A, its identity, to Bob so
   that Bob may select the appropriate credential for the sequel to the
   conversation. This leads to a slightly complemented version of AKEP2
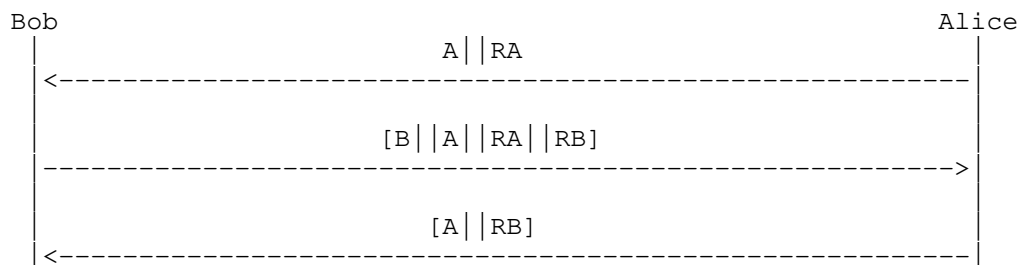   for EAP-PSK as depicted in Figure 5.

```
Bob                                                           Alice
 |                          A||RA                                |
 |<-------------------------------------------------------------|
 |                                                               |
 |                                                               |
 |                       [B||A||RA||RB]                          |
 |------------------------------------------------------------->|
 |                                                               |
 |                                                               |
 |                          [A||RB]                              |
 |<-------------------------------------------------------------|
```

                     Figure 5: Overview of AKEP2

   In AKEP2,

   o RA and RB are random numbers chosen respectively by Alice and Bob.

   o A and B are Alice's and Bob's respective identities. They allow
      Alice and Bob to retrieve the key that they have to use to run an
      authenticated key exchange between each other. They are also
      included in the protocol for cryptographic reasons.

o The MACs (see Section 1.3 for the notation "[]") are calculated
   using a dedicated key.

EAP-PSK instantiates this protocol with:

o The server as Alice and the peer as Bob.

o RA and RB as 16-byte random numbers, using Section 4.1 notations;
   this means RA=RAND_S and RB=RAND_P.

o A and B as Alice's and Bob's respective NAIs, using Section 4.1
   notations; this means A=ID_S and B=ID_P.

o The MAC algorithm as CMAC with AES-128 using AK and producing a
   tag length of 16 bytes.

o The modified counter mode of operation of AES-128 using KDK, to
   derive session keys as a result of this exchange.

CMAC was chosen as the MAC algorithm because it is capable of
handling arbitrary length messages, and its design is simple. It
also enjoys up-to-date review by the cryptographic community,
especially using provable security concepts. It has been recommended
by the NIST. For a detailed description of CMAC, please refer to
[8].

In AKEP2, the key exchange is "implicit": the session keys are
derived from RB. In EAP-PSK, the session keys are thus derived from
RAND_P by using KDK and the modified counter mode of operation of
AES-128 described in [5]. This mode was chosen because it is a
simple key derivation scheme that relies on a block cipher and has a
proof of its security. It is a length increasing function, i.e., it
expands one AES-128 input block into a longer t-block output, where
t>=2. The derivation of the session keys is shown in Figure 6.

```
+-------------------------+        +------------------------------+
|         RAND_P          |        |             KDK              |
|   Input Block (16 bytes)|        |  Key Derivation Key (16 bytes)|
+-------------------------+        +------------------------------+
             |                                    |
             v                                    v
+-----------------------------------------------------------------+
|                                                                 |
|                      Modified Counter Mode                      |
|                                                                 |
+-----------------------------------------------------------------+
        |                       |                       |
        v                       v                       v
+------------+   +----------------------+   +----------------------+
|    TEK     |   |         MSK          |   |         EMSK         |
| (16 bytes) |   |      (64 bytes)      |   |      (64 bytes)      |
+------------+   +----------------------+   +----------------------+
```
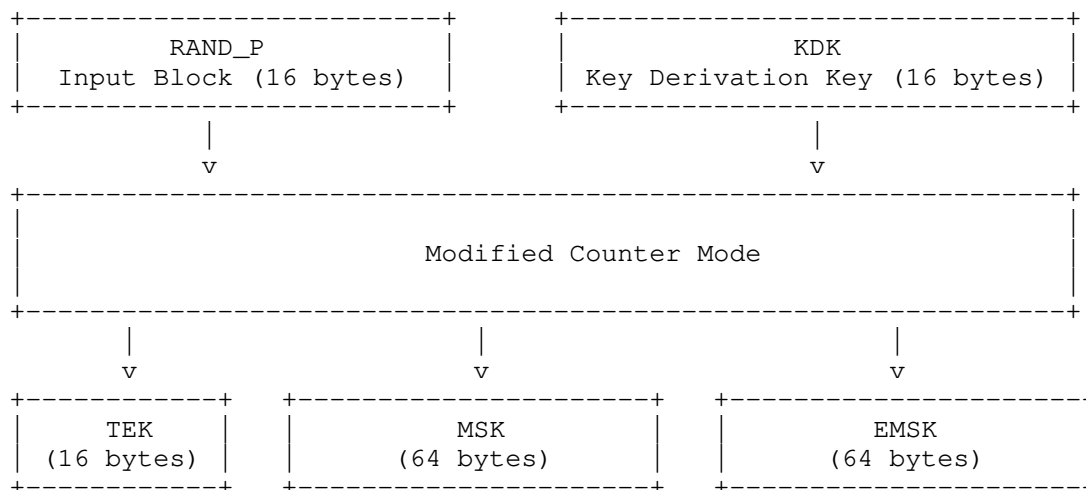
                 Figure 6: Derivation of the Session Keys

   The input to the derivation of the session keys is RAND_P.

   The outputs of the derivation of the session keys are:

   o   The 16-byte TEK (the first output block).

   o   The 64-byte MSK (the concatenation of the second to fifth output
       blocks).

   o   The 64-byte EMSK (the concatenation of the sixth to ninth output
       blocks).

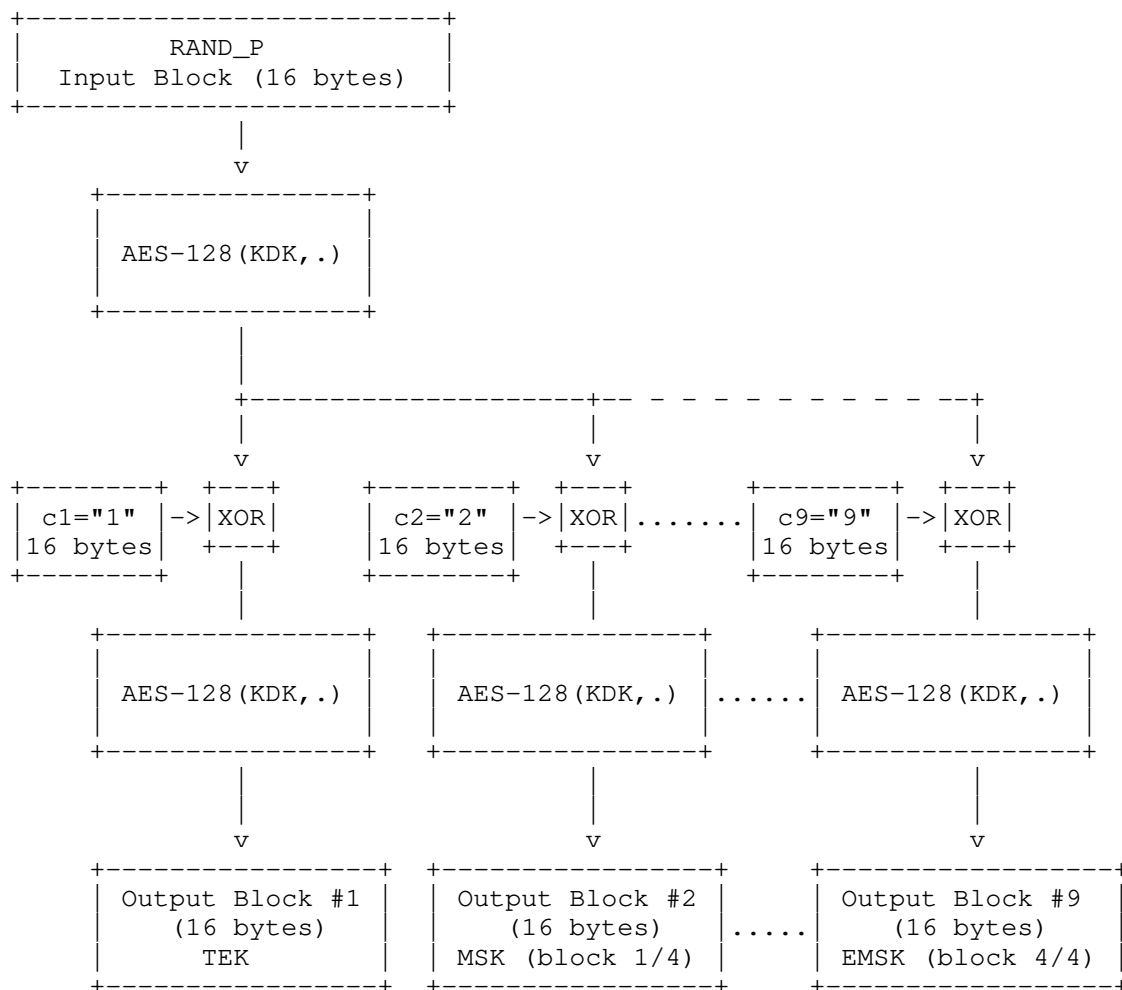   The details of the derivation of the session keys are shown in
   Figure 7.

```
          +------------------------+
          |         RAND_P         |
          |  Input Block (16 bytes)|
          +------------------------+
                      |
                      v
              +----------------+
              |                |
              | AES-128(KDK,.) |
              |                |
              +----------------+
                     |
                     |
              +------------------------+-- - - - - - - - - --+
              |                        |                     |
              v                        v                     v
  +--------+ +---+       +--------+ +---+          +--------+ +---+
  |c1="1"  |->|XOR|      |c2="2"  |->|XOR|.......  |c9="9"  |->|XOR|
  |16 bytes| +---+      |16 bytes| +---+          |16 bytes| +---+
  +--------+   |         +--------+   |            +--------+   |
               |                      |                         |
       +----------------+    +----------------+       +----------------+
       |                |    |                |       |                |
       | AES-128(KDK,.) |    | AES-128(KDK,.) |.......| AES-128(KDK,.) |
       |                |    |                |       |                |
       +----------------+    +----------------+       +----------------+
               |                      |                         |
               v                      v                         v
       +----------------+    +----------------+       +-----------------+
       | Output Block #1|    | Output Block #2|       | Output Block #9 |
       |    (16 bytes)  |    |    (16 bytes)  |.....  |    (16 bytes)   |
       |      TEK       |    |  MSK (block 1/4)|      | EMSK (block 4/4)|
       +----------------+    +----------------+       +-----------------+
```

                Figure 7: Derivation of the Session Keys in Details

   The counter values are set respectively to the first t integers (that
   is, c_i="i", with i=1 to 9).

   Keying material is sensitive information and should be handled
   accordingly (see Section 8.10 for further discussion).

3.3. The Protected Channel

   EAP-PSK provides a protected channel for both parties to communicate
   over, in case of a successful authentication. This protected channel
   is currently used to exchange protected result indications and may be
   used in the future to implement extensions.

   EAP-PSK uses the EAX mode of operation to provide this protected
   channel. For a detailed description of EAX, please refer to [4].
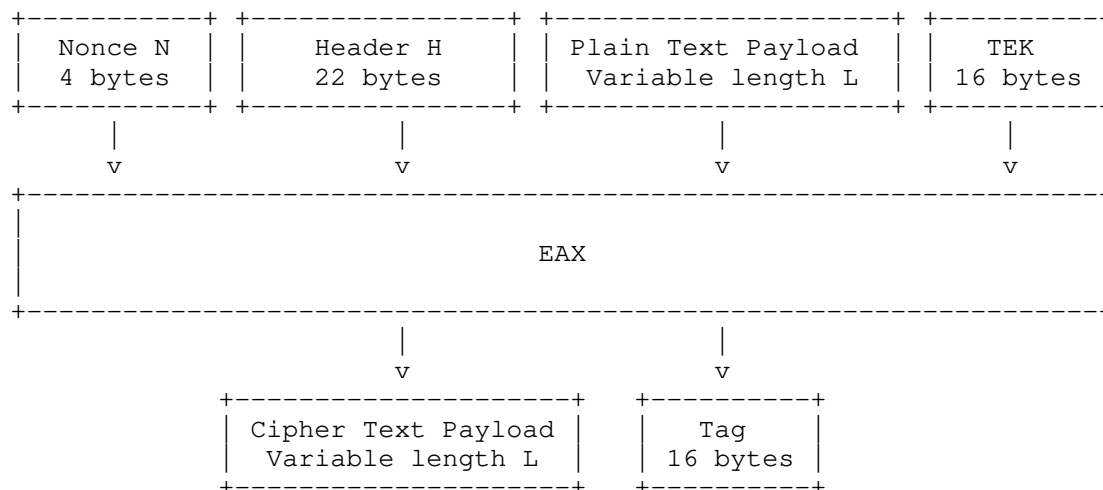   Figure 8 shows how EAX is used to implement EAP-PSK protected
   channel.

```
+----------+ +---------------+ +---------------------+ +----------+
|  Nonce N | |   Header H    | |  Plain Text Payload | |   TEK    |
|  4 bytes | |   22 bytes    | |  Variable length L  | | 16 bytes |
+----------+ +---------------+ +---------------------+ +----------+
     |               |                    |                  |
     v               v                    v                  v
+----------------------------------------------------------------+
|                                                                |
|                                                                |
|                             EAX                                |
|                                                                |
|                                                                |
+----------------------------------------------------------------+
                |                      |
                v                      v
     +---------------------+     +----------+
     | Cipher Text Payload |     |   Tag    |
     |  Variable length L  |     | 16 bytes |
     +---------------------+     +----------+
```

                   Figure 8: The Protected Channel

   This protected channel:

   o Provides replay protection.

   o Encrypts and authenticates a Plain Text Payload that becomes an
      Encrypted Payload. The Plain Text Payload must not exceed 960
      bytes; see Sections 5.3, 5.4, and 8.11.

   o Only authenticates a Header that is thus sent in clear.

   EAX is instantiated with AES-128 as the underlying block cipher.

   AES-128 is keyed with the TEK.

The nonce N is used to provide cryptographic security to the
encryption and data origin authentication as well as protection
replay. Indeed, N is a 4-byte sequence number starting from <0> that
is monotonically incremented at each EAP-PSK message within one EAP-
PSK dialog, except retransmissions, of course.

N was taken to be 4 bytes to avoid 16-byte arithmetic. Since EAX
uses a 16-byte nonce, N is padded with 96 zero bits for its high-
order bits.

For cryptographic reasons, N is not allowed to wrap around. In the
unlikely, yet possible, event of the server sending an EAP-PSK
message with N set to <2**32-2>, it must not send any further message
on this protected channel, which would cause to reusing the value 0.
Either the conversation is finished after the server receives the
EAP-PSK answer from the peer with N set to <2**32-1> and the server
proceeds (typically by sending an EAP-Success or Failure), or the
conversation is not finished and must then be aborted (a new EAP-PSK
dialog may subsequently be started to try again to authenticate).
Thus, the maximum number of messages that can be exchanged over the
same protected channel is 2**32 (which should not be a limitation in
practice, as this is approximately equal to 4 billion).

The Header H consists of the first 22 bytes of the EAP Request or
Response packet (i.e., the EAP Code, Identifier, Length, and Type
fields followed by the EAP-PSK Flags and RAND_S fields). Although it
may appear unorthodox that an upper layer (EAP-PSK) protects some
information of the lower layer (EAP), this was chosen to comply with
EAP recommendation (see Section 7.5. of [3]) and seems to be existing
practice at IETF (see, for instance, [35]).

The Plain Text Payload is the payload that is to be encrypted and
integrity protected. The Cipher Text Payload is the result of the
encryption of the Plain Text.

The Tag is a MAC that protects both the Header and the Plain Text
Payload. The verification of the Tag must only be done after a
successful verification of the Nonce for replay protection. If the
verification of the Tag succeeds, then the Encrypted Payload is
decrypted to recover the Plain Text Payload. If the verification of
the Tag fails, then no decryption is performed and this MAC failure
should be logged. The tag length is chosen to be 16 bytes for EAX
within EAP-PSK. This length is considered appropriate by the
cryptographic community.

EAX was mainly chosen because:

o It strongly relies on OMAC in its design and OMAC1, a variant of
   OMAC, had already been chosen in EAP-PSK for the authentication
   part (please remember that OMAC1 and CMAC are analogous).

o Its design is simple.

o It enjoys a security proof.

o It is free of any Intellectual Property Rights claims.

4. EAP-PSK Message Flows

   EAP-PSK may consist of two different types of message flows:

o The "standard authentication", which is:

   * Mandatory to implement.

   * Fully specified in this document.

   * The simpler type of message flow, which is expected to be used
      most frequently.

o The "extended authentication", which is:

   * Optional to implement (i.e., there are no mandatory
      extensions).

   * Partly specified in this document since it depends on
      extensions and none are currently specified, let alone in this
      document.

   * The type of message flow that should be used when extensions of
      EAP-PSK are needed by more sophisticated usage scenarios and
      are available.

   EAP-PSK introduces the concept of a session to facilitate its
   analysis and provide a cleaner interface to other layers. A session
   is a particular instance of an EAP-PSK dialog between two parties.
   This session is identified by a session identifier.

   In the first EAP-PSK message, the EAP server asserts its identity.
   Given that the EAP-Request/Identity and EAP-Response/Identity may not
   be assumed to have occurred prior to this sending and that the
   response included in EAP-Response/Identity (if this EAP Identity
   exchange takes place) may not contain the actual NAI the peer shall

use with EAP-PSK, this means that an EAP server implementing EAP-PSK
must use the same EAP server NAI for all EAP-PSK dialogs with any EAP
peer implementing EAP-PSK.

4.1. EAP-PSK Standard Authentication

EAP-PSK standard authentication is comprised of four messages, i.e.,
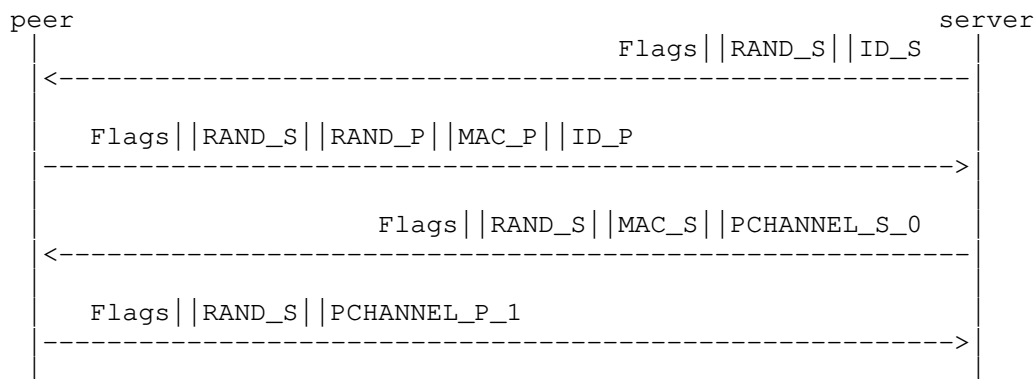two round-trips; see Figure 9.

```
peer                                                           server
 |                                    Flags||RAND_S||ID_S     |
 |<----------------------------------------------------------|
 |                                                           |
 |    Flags||RAND_S||RAND_P||MAC_P||ID_P                      |
 |---------------------------------------------------------->|
 |                                                           |
 |                         Flags||RAND_S||MAC_S||PCHANNEL_S_0 |
 |<----------------------------------------------------------|
 |                                                           |
 |    Flags||RAND_S||PCHANNEL_P_1                             |
 |---------------------------------------------------------->|
 |                                                           |
```

Figure 9: EAP-PSK Standard Authentication

o The first message is sent by the server to the peer to:

   * Send a 16-byte random challenge (RAND_S). RAND_S was called RA
      in Section 3.2

   * State its identity (ID_S). ID_S was denoted by A in
      Section 3.2.

o The second message is sent by the peer to the server to:

   * Send another 16-byte random challenge (RAND_P). RAND_P was
      called RB in Section 3.2

   * State its identity (ID_P). ID_P was denoted by B in
      Section 3.2.

   * Authenticate to the server by proving that it is able to
      compute a particular MAC (MAC_P), which is a function of the
      two challenges and AK:
      MAC_P = CMAC-AES-128(AK, ID_P||ID_S||RAND_S||RAND_P)

   o The third message is sent by the server to the peer to:

      * Authenticate to the peer by proving that it is able to compute
         another MAC (MAC_S), which is a function of the peer's
         challenge and AK:
         MAC_S = CMAC-AES-128(AK, ID_S||RAND_P)

      * Set up the protected channel (P_CHANNEL_S_0) to:

         + Confirm that it has derived session keys (at least the TEK).

         + Give a protected result indication of the authentication.

   o The fourth message is sent by the peer to the server to finish the
      setup of the protected channel (P_CHANNEL_P_1) to:

      * Confirm that it has derived session keys (at least the TEK).

      * Give a protected result indication of the authentication.

   The PCHANNEL_S_0 and PCHANNEL_P_1 fields of the third and fourth EAP-
   PSK messages contain a MAC-computed thanks to TEK that protects the
   integrity of the messages. For a detailed list of the fields of the
   messages that are integrity protected, please refer to Section 3.3.

   All EAP-PSK messages include a sort of header, which is comprised of
   two fields:

   o Flags, a 1-byte field that is currently only used to number EAP-
      PSK messages.

   o RAND_S, a 16-byte challenge sent by the server that is used as a
      session identifier.

   This standard message flow could be comprised of only three messages,
   like AKEP2, were it not the request/response nature of EAP that
   prevents the third message to be the last one. Since the fourth
   message is mandatory, EAP-PSK chose to take advantage of this and set
   up a protected channel.

   The standard message flow also includes a statement by the peer of
   its identity, in addition to the EAP-Response/Identity it may have
   sent. This behavior follows Section 5.1 of [3], which recommends
   that the EAP-Response/Identity be used primarily for routing purposes
   and selecting which EAP method to use, and therefore that EAP methods
   include a method-specific mechanism for obtaining the identity, so
   that they do not have to rely on the Identity Response.

When a party receives an EAP-PSK message, it checks that the message
is syntactically valid in accordance with the message formats defined
in Section 5. If the message is syntactically incorrect, then it is
silently discarded. Then it checks the cryptographic validity of
this message, i.e., it checks the MAC(s) as follows:

o If the received message is the first EAP-PSK message, there is no
   MAC to check as none is included in message 1.

o If the received message is the second EAP-PSK message, the
   validity of MAC_P is checked.

o If the received message is the third EAP-PSK message, the validity
   of MAC_S is checked and then the validity of the Tag included in
   P_CHANNEL_S_0 is checked. The validity checks must be done in
   this order to avoid unnecessarily deriving TEK, MSK, and EMSK in
   case MAC_S is invalid, meaning that mutual authentication has
   failed. Indeed, TEK is used to verify the validity of the Tag
   included in P_CHANNEL_S_0.

o If the received message is the fourth EAP-PSK message, the
   validity of the Tag included in P_CHANNEL_P_1 is checked.

If a validity check fails, the message is silently discarded. There
can be a counter to track the number of silently discarded messages
Section 8.8. If there is an encrypted payload in the message
(namely, in the PCHANNEL attribute), then the encrypted payload is
decrypted. Then, if the decrypted payload is syntactically
incorrect, the message is silently discarded.

4.2. EAP-PSK Extended Authentication

To remain simple and yet be extensible to meet future requirements,
EAP-PSK provides an extension mechanism within its protected channel:
the payload of the protected channel may contain an optional
extension field (EXT).

Figure 10 presents the message sequence for EAP-PSK extended
authentication.

Extended authentication MUST be supported, i.e., any EAP-PSK
implementation MUST support sending and reception of an EXT attribute
according to rules of operation described in Section 6. Yet,
although support of the EXT field is mandatory, there is no mandatory
extension type to support. This means that if a server engages in
EAP-PSK extended authentication, as only the server can start
extended authentication per Section 6, a peer will recognize the
attempt to start extended authentication through its EXT support. If

the peer does not support the particular extension type used by the
server, the peer will still be able to conclude the EAP-PSK dialog.

The mandatory support of the EXT field is dictated:

o To guarantee a robust behavior in the future where some peers
   might support some extensions and others not. All peers will thus
   be able to understand that an extended authentication is being
   attempted and indicate whether or not they support the extension
   that is tried.

o To ensure that all implementations will indeed be extensible.

No extension is currently defined.

At most, one extension may be run within a single EAP-PSK dialog:
there can neither be sequences of extensions nor interleaved
extensions. However, extensions may take a variable number of round-
trips to complete.

Only the server can start an extension and, if it does so, it must
start it in the first payload it sends over the protected channel.

```
peer                                                           server
 |                                     Flags||RAND_S||ID_S     |
 |<-----------------------------------------------------------|
 |                                                            |
 |    Flags||RAND_S||RAND_P||MAC_P||ID_P                       |
 |----------------------------------------------------------->|
 |                                                            |
 |                   Flags||RAND_S||MAC_S||PCHANNEL_S_0(EXT)   |
 |<-----------------------------------------------------------|
 |                                                            |
 |    Flags||RAND_S||PCHANNEL_P_1(EXT)                         |
 |----------------------------------------------------------->|
 |                                                            |
 .                                                            .
 .                                                            .
 .                                                            .
 |                   Flags||RAND_S||PCHANNEL_S_2i(EXT)         |
 |<-----------------------------------------------------------|
 |                                                            |
 |    Flags||RAND_S||PCHANNEL_P_2i+1(EXT)                      |
 |----------------------------------------------------------->|
 |                                                            |
```
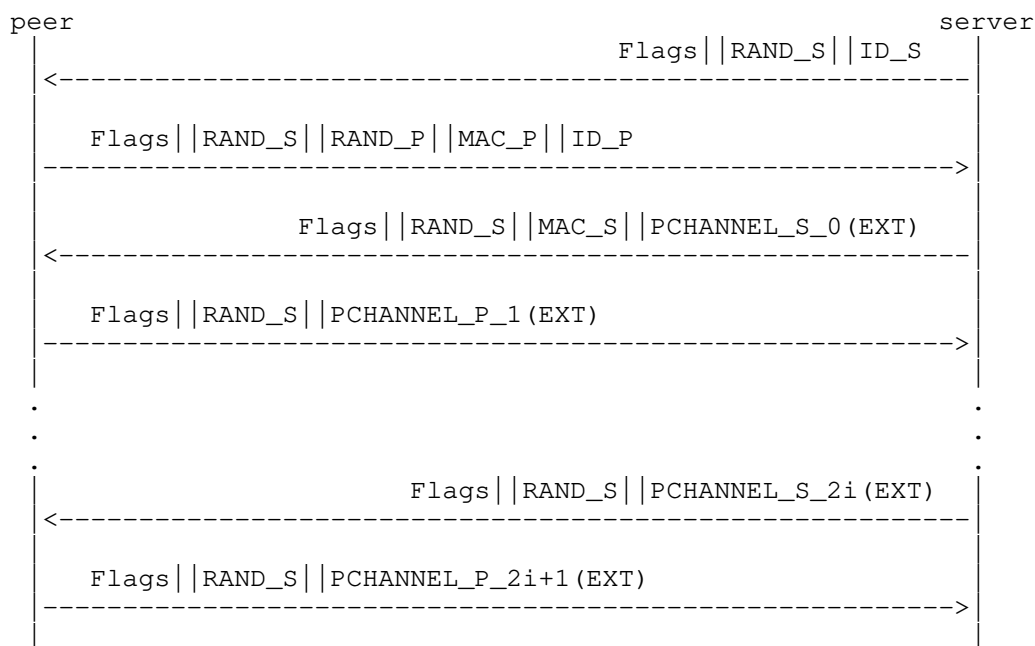
Figure 10: EAP-PSK Extended Authentication

Please refer to Section 6 for more details on how extended
authentication works.

The PCHANNEL_S_2j and PCHANNEL_P_2j+1 fields of the EAP-PSK messages
(where j varies from 0 to i) contain a MAC-computed thanks to TEK
that protects the integrity of the messages. For a detailed list of
the fields of the messages that are integrity protected, please refer
to Section 3.3.

When a party receives an EAP-PSK message, it checks that the message
is syntactically valid in accordance with the message formats defined
in Section 5. If the message is syntactically incorrect, then it is
silently discarded. Then it checks the cryptographic validity of
this message, i.e., it checks the MAC(s) as follows:

o If the received message is the first EAP-PSK message, there is no
   MAC to check as none is included in message 1.

o If the received message is the second EAP-PSK message, the
   validity of MAC_P is checked.

o If the received message is the third EAP-PSK message, the validity
   of MAC_S is checked and then the validity of the Tag included in
   P_CHANNEL_S_0 is checked. The validity checks must be done in
   this order to avoid unnecessarily deriving TEK, MSK, and EMSK in
   case MAC_S is invalid, meaning that mutual authentication has
   failed. Indeed, TEK is used to verify the validity of the Tag
   included in P_CHANNEL_S_0.

o If the received message is the fourth EAP-PSK message, the
   validity of the Tag included in P_CHANNEL_P_1 is checked.

o If the received message is an EAP-PSK message different from the
   first four ones, then validity of the Tag included in P_CHANNEL is
   checked.

If a validity check fails, the message is silently discarded. There
can be a counter to track the number of silently discarded messages
Section 8.8. If there is an encrypted payload in the message (namely
in the PCHANNEL attribute), then the encrypted payload is decrypted.
Then, if the decrypted payload is syntactically incorrect, the
message is silently discarded.

5. EAP-PSK Message Format

   For the sake of simplicity, EAP-PSK uses a fixed message format.
   There are four different types of EAP-PSK messages:

   o The first EAP-PSK message, which is sent by the server to the
      peer.

   o The second EAP-PSK message, which is sent by the peer to the
      server.

   o The third EAP-PSK message, which is sent by the server to the
      peer.

   o The fourth EAP-PSK message, which is sent by the peer to the
      server. This is also the type of message that the peer further
      sends to the server in case of an extended authentication. This
      is also essentially the type of message that the server further
      sends to the peer in case of an extended authentication: the only
      slight modification that occurs in this last case is the setting
      of the EAP Code to 1 instead of 2 in the other cases.

   For the sake of clarity, the whole EAP packet that encapsulates the
   EAP-PSK message (i.e., the EAP-PSK message plus its EAP headers) is
   depicted in Figures 11, 13, 14, and 18.

5.1. EAP-PSK First Message

   The first EAP-PSK message is sent by the server to the peer. It has
   the format presented in Figure 11.

```
 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=1     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Type EAP-PSK |     Flags     |                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                             |
+                                                             +
|                            RAND_S                           |
+                                                             +
|                                                             |
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
:                                                             :
:                             ID_S                            :
:                                                             :
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
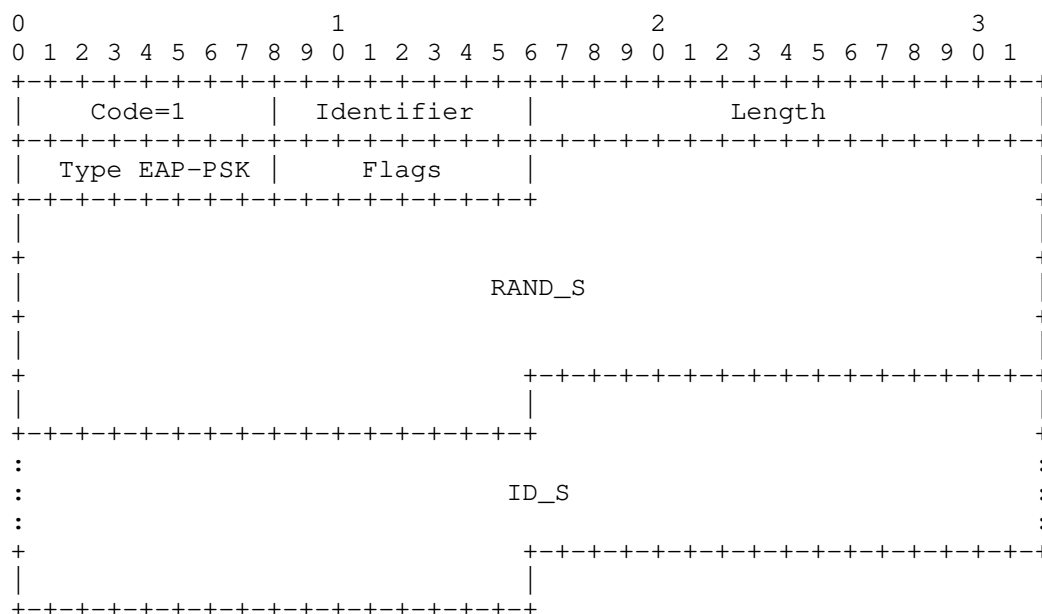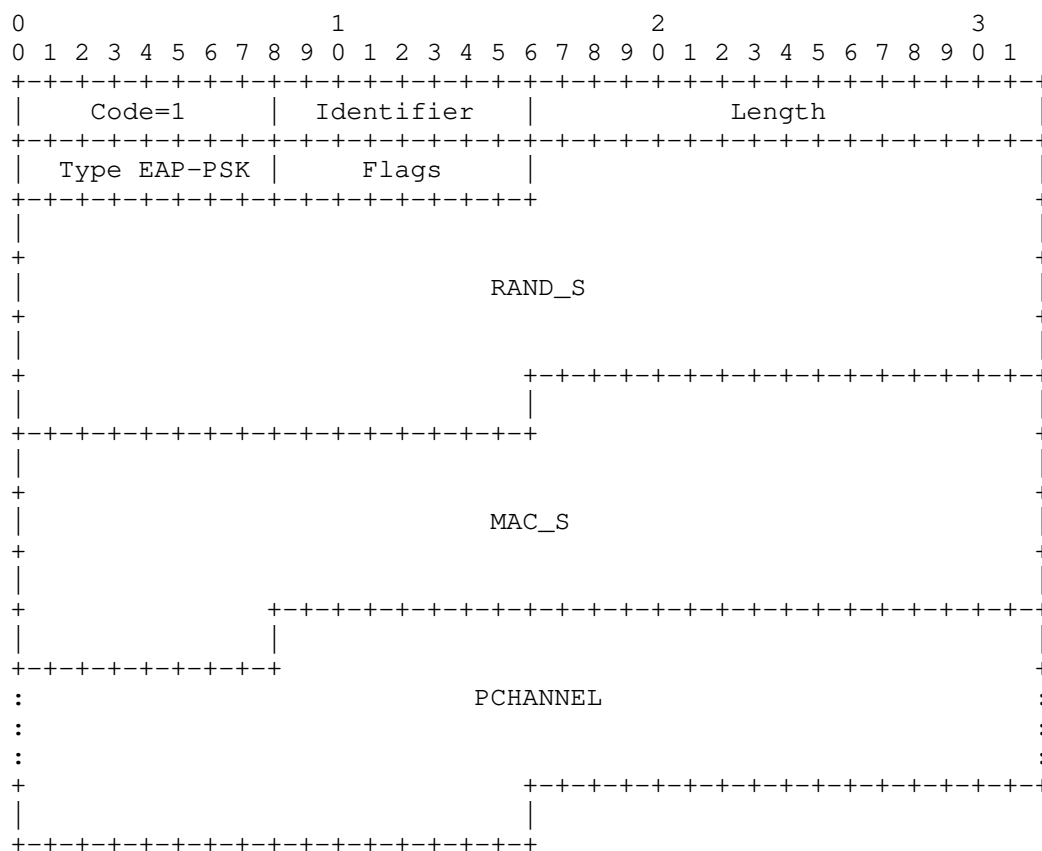
                  Figure 11: EAP-PSK First Message

   Since IANA has allocated EAP method type 47 for EAP-PSK, Type EAP-PSK
   for the first EAP-PSK message as well as any other EAP-PSK message
   MUST be 47.

   The first EAP-PSK message consists of:

   o A 1-byte Flags field

   o A 16-byte random number: RAND_S

   o A variable length field that conveys the server's NAI: ID_S. The
      length of this field is deduced from the EAP length field. The
      length of this NAI must not exceed 966 bytes. This restriction
      aims at avoiding fragmentation issues (see Section 8.11).

   The Flags field has the format presented in Figure 12.

```
0
0 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
| T | Reserved  |
+-+-+-+-+-+-+-+-+
```
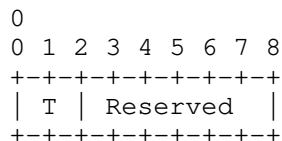
                        Figure 12: EAP-PSK Flags Field

The Flags field is comprised of two subfields:

o A 2-bit T subfield, which indicates the type of EAP-PSK message:

   * T=0 for the first EAP-PSK message presented in Section 5.1.

   * T=1 for the second EAP-PSK message presented in Section 5.2.

   * T=2 for the third EAP-PSK message presented in Section 5.3.

   * T=3 for the fourth EAP-PSK message presented in Section 5.4 and
      the subsequent EAP-PSK messages that may be exchanged during
      extended authentication.

o A 6-bit Reserved subfield that is set to zero on transmission and
   ignored on reception.

The PCHANNEL Nonce field N (see Section 5.3) is used to distinguish
between the different EAP-PSK messages that may be exchanged during
extended authentication that all have T set to 3, i.e., the fourth
EAP-PSK message and possibly the next ones.

5.2. EAP-PSK Second Message

   The second EAP-PSK message is sent by the peer to the server. It has
   the format presented in Figure 13.

```
 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=2     |   Identifier  |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Type EAP-PSK |     Flags     |                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                              +
|                                                              |
+                                                              +
|                             RAND_S                           |
+                                                              +
|                                                              |
+                                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                 |                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                            +
|                                                              |
+                                                              +
|                             RAND_P                           |
+                                                              +
|                                                              |
+                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 |                                            |
+-+-+-+-+-+-+-+-+-+                                            +
|                                                              |
+                                                              +
|                             MAC_P                            |
+                                                              +
|                                                              |
+                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 |                                            |
+-+-+-+-+-+-+-+-+-+                                            +
:                             ID_P                             :
:                                                              :
+                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
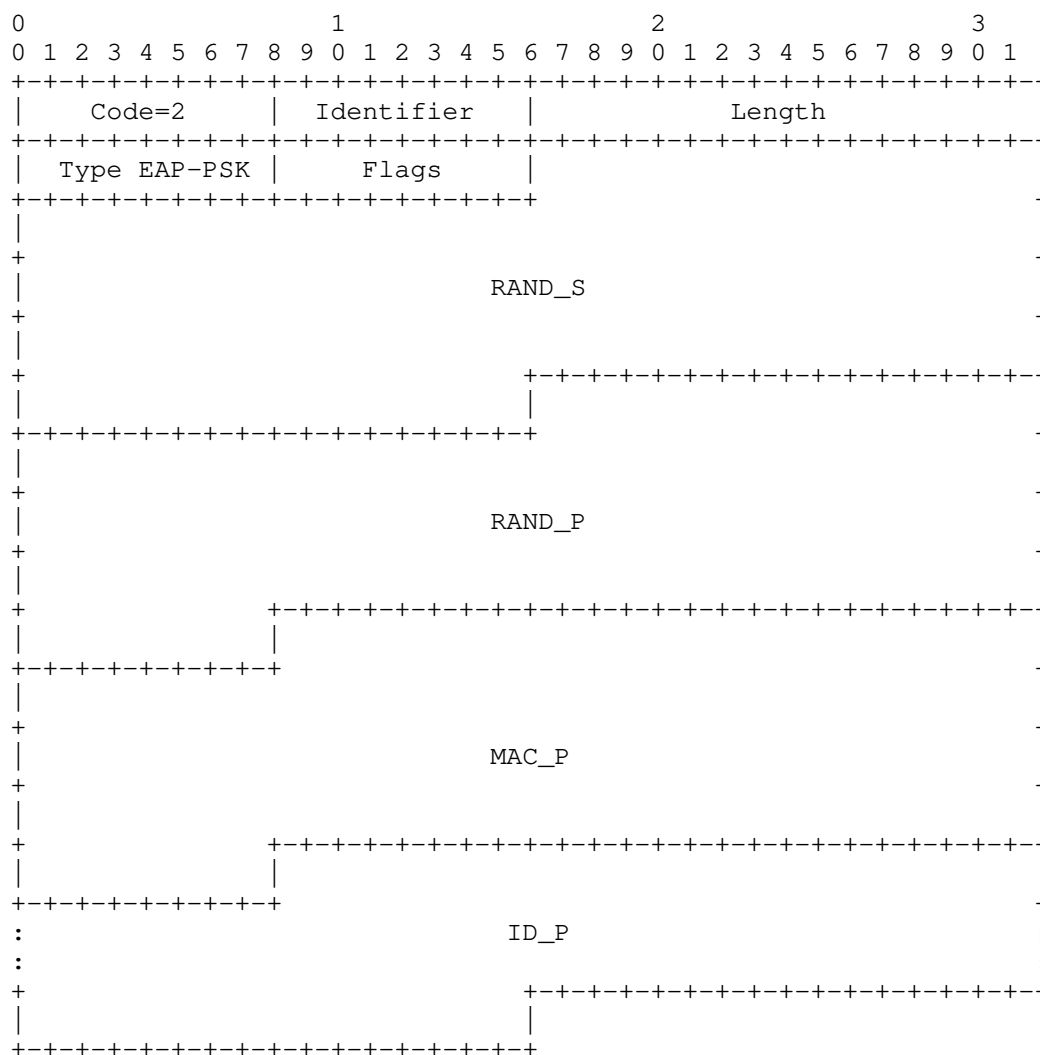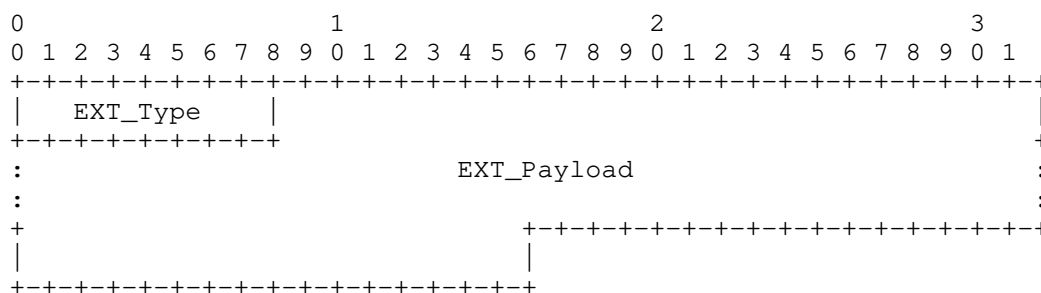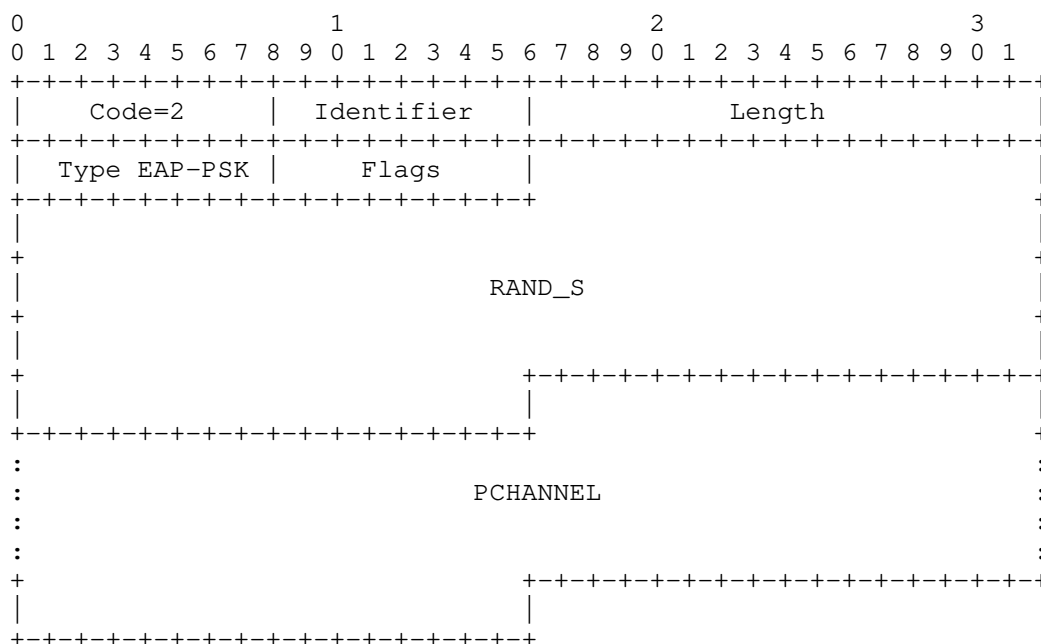
                  Figure 13: EAP-PSK Second Message

It consists of:

o A 1-byte Flags field

o The 16-byte random number sent by the server in the first EAP-PSK
   message (RAND_S) that serves as a session identifier

o A 16-byte random number: RAND_P

o A 16-byte MAC: MAC_P

o A variable length field that conveys the peer's NAI: ID_P. The
   length of this field is deduced from the EAP length field. The
   length of this NAI must not exceed 966 bytes. This restriction
   aims at avoiding fragmentation issues (see Section 8.11).

The Flags field format is presented in Figure 12.

5.3. EAP-PSK Third Message

   The third EAP-PSK message is sent by the server to the peer. It has
   the format presented in Figure 14.

```
 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=1     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type EAP-PSK  |    Flags      |                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                              +
|                                                              |
+                                                              +
|                           RAND_S                             |
+                                                              +
|                              +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              |                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                              +
|                                                              |
+                                                              +
|                           MAC_S                              |
+                                                              +
|              +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              |                                              |
+-+-+-+-+-+-+-+                                              +
:                           PCHANNEL                           :
:                                                              :
:                                                              :
+                              +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                   Figure 14: EAP-PSK Third Message

   It consists of:

   o A 1-byte Flags field

   o The 16-byte random number sent by the server in the first EAP-PSK
      message (RAND_S) that is used as a session identifier

   o A 16-byte MAC: MAC_S

   o A variable length field that constitutes the protected channel:
      PCHANNEL

The Flags field format is presented in Figure 12.

If there is no extension, i.e., if the authentication is standard,
the PCHANNEL field consists of:

o A 4-byte Nonce N (see Section 3.3).

o A 16-byte Tag (see Section 3.3).

o A 2-bit result indication flag R.

o A 1-bit extension flag E, which is set to 0.

o A 5-bit Reserved field, which is set to zero on emission and
   ignored on reception.

R, E, and Reserved are sent encrypted by the protected channel (see
Section 3.3).

If there is no extension, PCHANNEL has the format presented in
Figure 15 (where R, E, and Reserved are presented in the clear for
the sake of clarity, although in reality they are sent encrypted).

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Nonce                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                             Tag                               |
+                                                               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| R |0| Reserved|
+-+-+-+-+-+-+-+-+
```
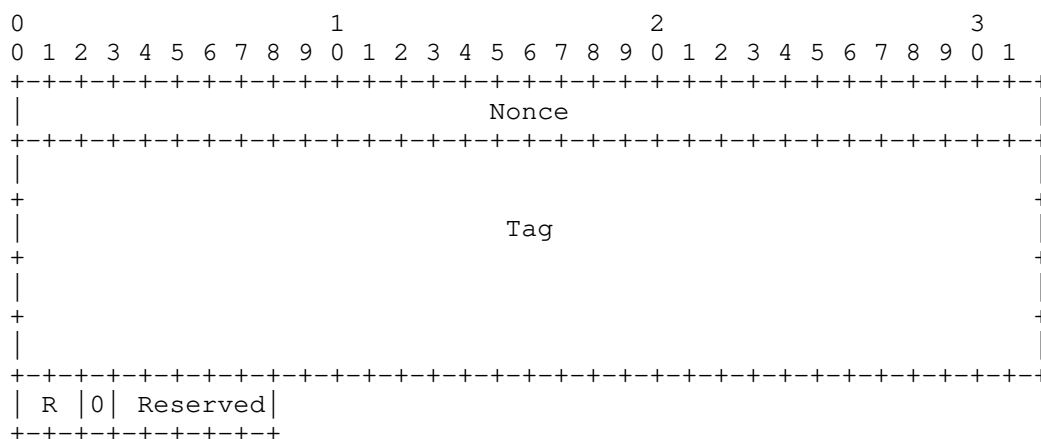
                Figure 15: The PCHANNEL Field with E=0

If there is an extension, i.e., if the authentication is extended,
the PCHANNEL field consists of:

o A 4-byte Nonce N (see Section 3.3).

o A 16-byte Tag (see Section 3.3).

o A 2-bit result indication flag R.

o A 1-bit extension flag E, which is set to 1.

o A 5-bit Reserved field, which is set to zero on emission and
   ignored on reception.

o A variable length EXT field.

R, E, Reserved, and EXT are sent encrypted by the protected channel
(see Section 3.3).

If there is an extension, PCHANNEL has the format presented in
Figure 16 where R, E, Reserved and EXT are presented in the clear for
the sake of clarity, although in reality they are sent encrypted).

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Nonce                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                             Tag                               |
+                                                               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| R |1| Reserved|                                               |
+-+-+-+-+-+-+-+                                                 +
:                             EXT                               :
:                                                               :
+                           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
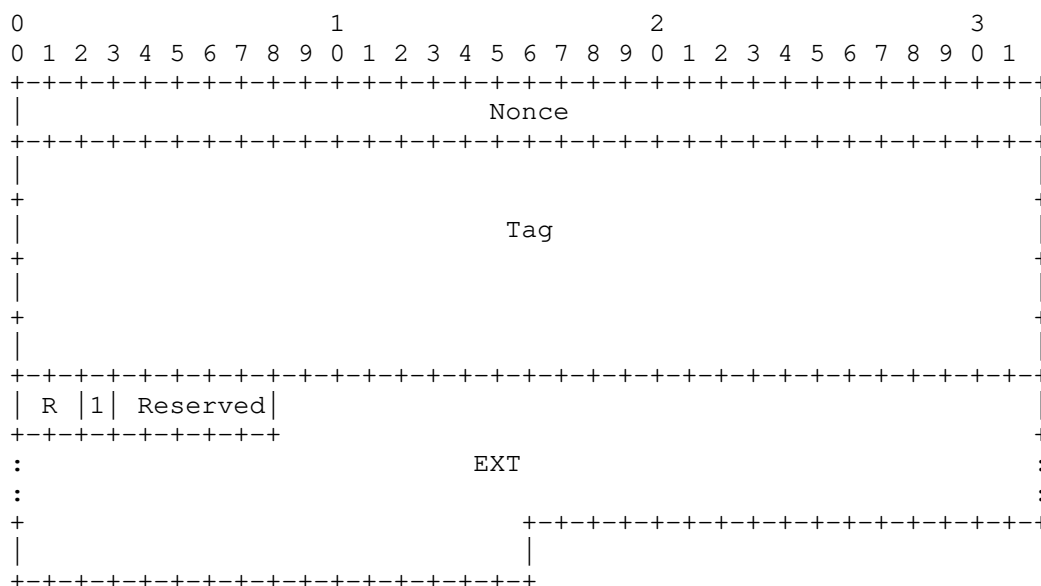
                   Figure 16: The PCHANNEL Field with E=1

This EXT field is split in two subfields:

o The EXT_Type subfield, which indicates the type of the extension

o The EXT_Payload subfield, which consists of the payload of the
   extension. The EXT_Payload length is derived from the EAP Length
   field. EXT_Payload must have a bit length that is a multiple of 8
   bits and must not exceed 960 bytes. The latter restriction aims

at avoiding fragmentation issues (see Section 8.11), whereas the
former comes from the EAP length being specified in bytes.

The format of the EXT field is presented in Figure 17.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    EXT_Type   |                                               |
+-+-+-+-+-+-+-+-+                                               +
:                         EXT_Payload                           :
:                                                               :
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 17: The EXT Field

5.4. EAP-PSK Fourth Message

The fourth EAP-PSK message is sent by the peer to the server. It has
the format presented in Figure 18.

```
 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=2     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Type EAP-PSK |    Flags      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
+                                                               +
|                             RAND_S                            |
+                                                               +
|                                                               |
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
:                                                               :
:                            PCHANNEL                           :
:                                                               :
:                                                               :
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18: EAP-PSK Fourth Message

It consists of:

o A 1-byte Flags field

o The 16-byte random number sent by the server in the first EAP-PSK
   message (RAND_S) that is used as a session identifier

o A variable length field that constitutes the protected channel:
   PCHANNEL

The Flags field format is presented in Figure 12.

The PCHANNEL field has the following structure, which was already
described in Section 5.3.

If there is no extension, i.e., if the authentication is standard,
the PCHANNEL field consists of:

o A 4-byte Nonce N (see Section 3.3).

o A 16-byte Tag (see Section 3.3).

o A 2-bit result indication flag R.

o A 1-bit extension flag E, which is set to 0.

o A 5-bit Reserved field, which is set to zero on emission and
   ignored on reception.

R, E, and Reserved are sent encrypted by the protected channel (see
Section 3.3).

If there is no extension, PCHANNEL has the format presented in
Figure 15.

If there is an extension, i.e., if the authentication is extended,
the PCHANNEL field consists of:

o A 4-byte Nonce N (see Section 3.3).

o A 16-byte Tag (see Section 3.3).

o A 2-bit result indication flag R.

o A 1-bit extension flag E, which is set to 1.

o A 5-bit Reserved field, which is set to zero on emission and
   ignored on reception.

   o A variable length EXT field.

   R, E, Reserved, and EXT are sent encrypted by the protected channel
   (see Section 3.3).

   If there is an extension, PCHANNEL has the format presented in
   Figure 16.

   This EXT field is split in two subfields:

   o The EXT_Type subfield, which indicates the type of the extension

   o The EXT_Payload subfield, which consists of the payload of the
      extension. The EXT_Payload length is derived from the EAP Length
      field. EXT_Payload must have a bit length that is a multiple of 8
      bits and must not exceed 960 bytes. The latter restriction aims
      at avoiding fragmentation issues (see Section 8.11).

   The format of the EXT field is presented in Figure 17.

6. Rules of Operation for the EAP-PSK Protected Channel

   In this section, the rules of operation of the EAP-PSK protected
   channel are presented:

   o How protected result indications are implemented.

   o How an extended authentication works in details.

6.1. Protected Result Indications

   The R flag of the PCHANNEL field in the third and fourth types of
   EAP-PSK messages is used to provide result indications.

   Since this 2-bit flag is communicated over the protected channel, it
   is:

   o Encrypted so that only the peer and the server can know its value.

   o Integrity-protected so that it cannot be modified by an attacker
      without the peer or the server detecting this modification.

   o Protected against replays.

   This 2-bit R flag can take the following values:

   o 01 to mean CONT

   o 10 to mean DONE_SUCCESS

   o 11 to mean DONE_FAILURE

   The peer and the server each remember some information about both the
   values of R that they have sent and the values of R they have
   received. It is the conjunction of both sent and received R values
   that indicate the success or the failure of the EAP-PSK dialog.

   In the case of a standard authentication, the following values of R
   should be exchanged:

   o Either the server sends a DONE_SUCCESS in the PCHANNEL of the
      third EAP-PSK message, to which the peer replies with a
      DONE_SUCCESS in the PCHANNEL of the fourth EAP-PSK message, which
      successfully ends the EAP-PSK dialog.

   o Or the server sends a DONE_FAILURE in the PCHANNEL of the third
      EAP-PSK message, to which the peer replies with a DONE_FAILURE in
      the PCHANNEL of the fourth EAP-PSK message, which unsuccessfully
      ends the EAP-PSK dialog.

   In the case of an extended authentication, more complex exchanges may
   occur, which is why the CONT value was introduced.

   The rules of operation for each value that R may take are detailed
   below.

6.1.1. CONT

   The server and the peer each initialize the values of R they intend
   to send and receive as CONT.

   Here CONT stands for "Continue". It indicates that the EAP-PSK
   dialog is not yet successful and that the party sending it wants to
   continue the dialog to try and reach success.

   Indeed, although the peer and the server must have successfully
   authenticated each other, thanks to MAC_P and MAC_S, before they
   start communicating over the protected channel, the EAP-PSK dialog
   may not yet be deemed successful after this mutual authentication
   because of authorization issues. For instance, a prepaid customer of
   a wireless Hot-Spot might have successfully authenticated but has to
   refill its account, e.g., with a credit card transaction over the
   protected channel, before it is authorized.

6.1.2. DONE_SUCCESS

   DONE_SUCCESS indicates that the party that sent it deems the EAP-PSK
   dialog successful and therefore proposes to end this dialog.

   Once the server has sent a DONE_SUCCESS, it must keep sending this
   value for R.

   The peer must first receive a DONE_SUCCESS from the server before it
   is allowed to send a DONE_SUCCESS.

   After the peer has received a DONE_SUCCESS from the server, it may:

   o Send a CONT to the server if it has not reached success on its
      side. The server that receives a CONT should continue the EAP-PSK
      dialog (see Section 8.2 for some discussion on the security
      implications of this).

   o Send a DONE_SUCCESS to the server, which will end the EAP-PSK
      dialog with success.

   o Send a DONE_FAILURE to the server, which will end the EAP-PSK
      dialog with failure.

6.1.3. DONE_FAILURE

   DONE_FAILURE indicates that the party that sent it deems the EAP-PSK
   dialog unsuccessful and proposes to end this dialog because nothing
   will make it change its mind.

   If the server is the first to send a DONE_FAILURE, then the peer that
   receives this DONE_FAILURE must reply with a DONE_FAILURE and fail,
   which ends the EAP-PSK dialog.

   If the peer is the first to send a DONE_FAILURE, then the server that
   receives this DONE_FAILURE must immediately end this EAP-PSK dialog
   without sending any further EAP-PSK message, and fail.

6.2. Extended Authentication

   An extended authentication can only be started by the server.

   Exactly one extension (identified by the EXT_Type subfield of the EXT
   field) must be run during an EAP-PSK extended authentication dialog.

   The extension is run over the protected channel: it can assume
   confidentiality, integrity, and replay protection.

To start an extended authentication, the server sets the PCHANNEL E
flag to 1 and includes the EXT_Payload of the extension it has
chosen.

Since EAP-PSK does not provide fragmentation, the extension must not
send an EXT_Payload larger than 960 bytes, which corresponds to the
1020-byte EAP MTU that may minimally be assumed (see [3]).

Moreover, an extension must not send an empty EXT_Payload (because
this has a particular meaning for EAP-PSK; see below).

When the peer receives the third EAP-PSK message with the E flag set
to 1, it checks whether it is able to process the proposed extension.

If the peer is not able to process the proposed extension, i.e., it
does not recognize the EXT_Type of the proposed extension, it sets
E=1 in its reply (the fourth EAP-PSK message) and include an EXT
field of the same EXT_Type but with an empty EXT_Payload.

Depending on the values taken by the R flags, the EAP-PSK dialog may:

o End

   * If the peer's policy mandates that it fails in the case of an
      unrecognized extension, it sends a DONE_FAILURE in the fourth
      EAP-PSK message.

   * If the server has sent a DONE_SUCCESS in the third EAP-PSK
      message, and the peer's policy authorizes it to succeed even if
      the extension is not recognized, the peer sends a DONE_SUCCESS.

o Continue for exactly one round-trip; namely, in case the server
   has sent a CONT in the third EAP-PSK message and the peer's policy
   authorizes it to succeed even if the extension is not recognized,
   the peer replies with a CONT in the fourth EAP-PSK message. The
   server must then, depending on its policy, send either a
   DONE_SUCCESS or a DONE_FAILURE to the peer in the fifth EAP-PSK
   message. If the server sent a DONE_SUCCESS in the fifth EAP-PSK
   message, the peer must send a DONE_SUCCESS in the sixth EAP-PSK
   message. All these messages must have the E flag set to 1 with an
   EXT field with the EXT_Type of the extension that was proposed and
   an empty EXT_Payload (this behavior was chosen to simplify
   implementations).

If the peer is able to process the proposed extension, then it does
so. In this case, the extension must be aware of the R values sent
and received and able to propose to update them. All the subsequent
messages exchanged between the peer and the server must have the E

flag set to 1 with an EXT field of the EXT_Type of the extension that
was proposed and a non-empty EXT_Payload.

7. IANA Considerations

    This section provides guidance to the IANA regarding registration of
    values related to the EAP-PSK protocol, in accordance with [6].

    The following terms are used here with the meanings defined in [6]:
    "name space" and "registration".

    The following policies are used here with the meanings defined in
    [6]: "Expert Review" and "Specification Required".

    This document introduces one new Internet Assigned Numbers Authority
    (IANA) consideration: there is one name space in EAP-PSK that
    requires registration: the EXT_Type values (see Section 5.3 and
    Section 5.4).

    For registration requests where a Designated Expert should be
    consulted, the responsible IETF Area Director should appoint the
    Designated Expert. The intention is that any allocation will be
    accompanied by a published RFC. But in order to allow for the
    allocation of values prior to the RFC being approved for publication,
    the Designated Expert can approve allocations once it seems clear
    that an RFC will be published. The Designated Expert will post a
    request to the EAP WG mailing list (or a successor designated by the
    Area Director) for comment and review, including an Internet-Draft.
    Before a period of 30 days has passed, the Designated Expert will
    either approve or deny the registration request and publish a notice
    of the decision to the EAP WG mailing list or its successor, as well
    as informing IANA. A denial notice must be justified by an
    explanation and, in the cases where it is possible, concrete
    suggestions on how the request can be modified so as to become
    acceptable.

7.1. Allocation of an EAP-Request/Response Type for EAP-PSK

    IANA allocated a new EAP Type for EAP-PSK.

7.2. Allocation of EXT Type Numbers

    EAP-PSK is not intended as a general-purpose protocol, and
    allocations of EXT_Type should not be made for purposes unrelated to
    authentication, authorization, and accounting.

    EXT_Type numbers have a range from 1 to 255.

EXT_Type 255 has been allocated for Experimental use.

EXT_Type 1-254 may be allocated on the advice of a Designated Expert, with Specification Required.

8. Security Considerations

[3] highlights several attacks that are possible against EAP, as EAP does not provide any robust security mechanism.

This section discusses the claimed security properties of EAP-PSK as well as vulnerabilities and security recommendations in the threat model of [3].

8.1. Mutual Authentication

EAP-PSK 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 authentication protocol that inspired EAP-PSK, AKEP2, enjoys a security proof in the provable security paradigm; see [14].

The MAC algorithm used in the instantiation of AKEP2 within EAP-PSK, CMAC, also enjoys a security proof in the provable security paradigm; see [29]. A tag length of 16 bytes for CMAC is currently deemed appropriate by the cryptographic community for entity authentication.

The underlying block cipher used, AES-128, is widely believed to be a secure block cipher.

Finally, the key used for mutual authentication, AK, is only used for that purpose, which makes this part cryptographically independent of the other parts of the protocol.

EAP-PSK provides mutual authentication if it is based on a pairwise PSK of sufficient strength. If the PSK is not pairwise or not sufficiently strong, then it does not provide authentication. In this way, EAP-PSK is no different than other authentication protocols based on Pre-Shared Keys.

8.2. Protected Result Indications

   EAP-PSK provides protected result indications thanks to its 2-bit R
   flag (see Section 6.1). This 2-bit R flag is protected because it is
   encrypted and integrity protected by the EAX mode of operation; see
   Section 3.3.

   Care may be taken against Byzantine failures, that is to say, for
   instance, when a peer tries to force a server to engage in a never-
   ending conversation. This could, for example, be done by a peer that
   keeps sending a CONT after it has received a DONE_SUCCESS from the
   server. A policy may limit the number of rounds in an EAP-PSK
   extended authentication to mitigate this threat, which is outside our
   threat model.

   It should also be noted that the cryptographic protection of the
   result indications does not prevent message deletion.

   For instance, let us consider a scenario in which:

   o A server sends a DONE_SUCCESS to a peer.

   o The peer replies with a DONE_SUCCESS.

   In the case that the last message from the peer is intercepted, and
   an EAP Success is sent to the peer before any retransmission from the
   server reaches it, or the retransmissions from the server are also
   deleted, the peer will believe that it has successfully authenticated
   to the server while the server will fail.

   This behavior is well known (see, e.g., [23]) and in a sense
   unavoidable. There is a trade-off between efficiency and the "level"
   of information sharing that is attainable. EAP-PSK specified a
   single round-trip of DONE_SUCCESS because it is believed that:

   o If there is an adversary capable of disrupting the communication
      channel, it can do so whenever it wants (be it after 1 or 10
      round-trips or even during data communication).

   o Other layers/applications will generally start by doing a specific
      key exchange and confirmation procedure using the keys derived by
      EAP-PSK. This is typically done by IEEE 802.11i "four-way
      handshake". In case the error is not detected by EAP-PSK, it
      should be detected then (please note, however, that it is bad
      practice to rely on an external mechanism to ensure
      synchronization, unless this is an explicit property of the
      external mechanism).

8.3. Integrity Protection

   EAP-PSK provides integrity protection thanks to the Tag of its
   protected channel (see Section 3.3).

   EAP-PSK provides integrity protection if it is based on a pairwise
   PSK of sufficient strength. If the PSK is not pairwise or not
   sufficiently strong, then it does not provide authentication. In
   this way, it is no different than other authentication protocols
   based on Pre-Shared Keys.

8.4. Replay Protection

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

   EAP-PSK provides replay protection during the conversation of the
   protected channel thanks to the Nonce N of its protected channel (see
   Section 3.3). This nonce is initialized to 0 by the server and
   monotonically incremented by one by the party that receives a valid
   EAP-PSK message. For instance, after receiving from the server a
   valid EAP-PSK message with Nonce set to x, the peer will answer with
   an EAP-PSK message with Nonce set to x+1 and wait for an EAP-PSK
   message with Nonce set to x+2. A retransmission of the server's
   message with Nonce set to x would cause the peer EAP layer to resend
   the message in which Nonce was set to x+1, which would be transparent
   to the EAP-PSK layer.

   The EAP peer must check that the Nonce is indeed initialized to 0 by
   the server.

8.5. Reflection Attacks

   EAP-PSK provides protection against reflection attacks in case of an
   extended authentication because:

   o It integrity protects the EAP header (which contains the
      indication Request/Response.

   o It includes two separate spaces for the Nonces: the EAP server
      only receives messages with odd nonces, whereas the EAP peer only
      receives messages with even nonces.

8.6. Dictionary Attacks

   Because EAP-PSK is not a password protocol, it is not vulnerable to
   dictionary attacks.

   Indeed, the PSK used by EAP-PSK must not be derived from a password.
   Derivation of the PSK from a password may lead to dictionary attacks.

   However, using a 16-byte PSK has:

   o Ergonomic impacts: some people may find it cumbersome to manually
      provision a 16-byte PSK.

   o Deployment impacts: some people may want to reuse existing
      credential databases that contain passwords and not PSKs.

   Because people will probably not heed the warning not to use
   passwords, guidance to derive a PSK from a password is provided in
   Appendix A. The method proposed in Appendix A only tries to make
   dictionary attacks harder. It does not eliminate them.

   However, it does not cause a fatal error if passwords are used
   instead of PSKs: people rarely use password-derived certificates, so
   why should they do so for shared keys?

8.7. Key Derivation

   EAP-PSK supports key derivation.

   The key hierarchy is specified in Section 2.1.

   The mechanism used for key derivation is the modified counter mode.

   The instantiation of the modified counter in EAP-PSK complies with
   the conditions stated in [5] so that the security proof for this mode
   holds.

   The underlying block cipher used, AES-128, is widely believed to be a
   secure block cipher.

   A first key derivation occurs to calculate AK and KDK from the PSK:
   it is called the key setup (see Section 3.1). It uses the PSK as the
   key to the modified counter mode. Thus, AK and KDK are believed to
   be cryptographically separated and computable only to those who have
   knowledge of the PSK.

   A second key derivation occurs to derive session keys, namely, the
   TEK, MSK, and EMSK (see Section 3.2). It uses KDK as the key to the
   modified counter mode.

   The protocol design explicitly assumes that neither AK nor KDK are
   shared beyond the two parties utilizing them. AK loses its efficacy
   to mutually authenticate the peer and server with each other when it
   is shared. Similarly, the derived TEK, MSK, and EMSK lose their
   value when KDK is shared with a third party.

   It should be emphasized that the peer has control of the session keys
   derived by EAP-PSK. In particular, it can easily choose the random
   number it sends in EAP-PSK so that one of the nine derived 16-byte
   key blocks (see Section 2.1) takes a pre-specified value.

   It was chosen not to prevent this control of the session keys by the
   peer because:

   o Preventing it would have added some complexity to the protocol
      (typically, the inclusion of a one-way mode of operation of AES in
      the key derivation part).

   o It is believed that the peer won't try to force the server to use
      some pre-specified value for the session keys. Such an attack is
      outside the threat model and seems to have little value compared
      to a peer sharing its PSK.

   However, this is not the behavior recommended by EAP in Section 7.10
   of [3].

   Since deriving the session keys requires some cryptographic
   computations, it is recommended that the session keys be derived only
   once authentication has succeeded (i.e., once the server has
   successfully verified MAC_P for the server side, and once the peer
   has successfully verified MAC_S for the peer side).

   It is recommended to take great care in implementations, so that
   derived keys are not made available if the EAP-PSK dialog fails
   (e.g., ends with DONE_FAILURE).

   The TEK must not be made available to anyone except to the current
   EAP-PSK dialog.

8.8. Denial-of-Service Resistance

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

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

   It is worth noting that the server has to do a cryptographic
   calculation and maintain some state when it engages in an EAP-PSK
   conversation, namely, generate and remember the 16-byte RAND_S.
   However, this should not lead to resource exhaustion as this state
   and the associated computation are fairly lightweight.

   Please note that both the peer and the server must commit to their
   RAND_S and RAND_P to protect their partners from flooding attacks.

   It is recommended that EAP-PSK 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 that would delay
   his or her 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-PSK or to the peer and the
   server to specify the maximum number of failed cryptographic checks
   that are allowed. For instance, does the reception of a bogus MAC_P
   in the second EAP-PSK message cause a fatal error or is it discarded
   to continue waiting for the valid response of the valid peer? There
   is a trade-off between possibly allowing multiple tentative forgeries
   and allowing a direct DoS (in case the first error is fatal).

   For the sake of simplicity and denial-of-service resilience, EAP-PSK
   has chosen not to include any error messages. Hence, an "invalid"
   EAP-PSK message is silently discarded. Although this makes
   interoperability testing and debugging harder, this leads to simpler
   implementations and does not open any venue for denial-of-service
   attacks.

8.9. Session Independence

   Thanks to its key derivation mechanisms, EAP-PSK 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_P and RAND_S are random is central for the
security of EAP-PSK in general and session independence in
particular.

8.10. Exposition of the PSK

EAP-PSK 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: compromise of the PSK leads to "full" compromise of
future sessions.

EAP-PSK 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, whose choice is outside the
scope of this document. The PSK used by EAP-PSK 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-PSK must be cryptographically separated from keys
used by other protocols, otherwise the security of EAP-PSK may be
compromised. It is a rule of thumb in cryptography to use different
keys for different applications.

8.11. Fragmentation

EAP-PSK does not support fragmentation and reassembly.

Indeed, the largest EAP-PSK frame is at most 1015 bytes long,
because:

o The maximum length for the peer NAI identity used in EAP-PSK is
   966 bytes (see Section 5.2). This should not be a limitation in
   practice (see Section 2.2 of [2] for more considerations on NAI
   length).

o The maximum length for the EXT_Payload field used in EAP-PSK is
   960 bytes (see Section 5.3 and Section 5.4).

Per Section 3.1 of [3], the lower layers over which EAP may be run
are assumed to have an EAP MTU of 1020 bytes or greater. Since the
EAP header is 5 bytes long, supporting fragmentation for EAP-PSK is
unnecessary.

Extensions that require sending a payload larger than 960 bytes
should provide their own fragmentation and reassembly mechanism.

8.12. Channel Binding

   EAP-PSK does not provide channel binding as this feature is still
   very much a work in progress (see [13]).

   However, it should be easy to add it to EAP-PSK as an extension (see
   Section 4.2).

8.13. Fast Reconnect

   EAP-PSK does not provide any fast reconnect capability.

   Indeed, as noted, for instance, in [15], mutual authentication
   (without counters or timestamps) requires three exchanges, thus four
   exchanges in EAP since any EAP-Request must be answered to by an EAP-
   Response.

   Since this minimum bound is already reached in EAP-PSK standard
   authentication, there is no way the number of round-trips used within
   EAP-PSK can be reduced without using timestamps or counters.
   Timestamps and counters were deliberately avoided for the sake of
   simplicity and security (e.g., synchronization issues).

8.14. Identity Protection

   Since it was chosen to restrict to a single cryptographic primitive
   from symmetric cryptography, namely, the block cipher AES-128, it
   appears that it is not possible to provide "reasonable" identity
   protection without failing to meet the simplicity goal.

   Hereafter is an informal discussion of what is meant by identity
   protection and the rationale behind the requirement of identity
   protection. For some complementary discussion, refer to [37].

   Identity protection basically means preventing the disclosure of the
   identities of the communicating parties over the network, which is
   quite contradictory to authentication. There are two levels of
   identity protection: protection against passive attackers and
   protection against active eavesdroppers.

   As explained in [37], "a common example [for identity protection] is
   the case of mobile devices wishing to prevent an attacker from
   correlating their (changing) location with the logical identity of
   the device (or user)".

   If only symmetric cryptography is used, only a weak form of identity
   protection may be offered, namely, pseudonym management. In other
   words, the peer and the server agree on pseudonyms that they use to

identify each other and usually change them periodically, possibly in
a protected way so that an attacker cannot learn new pseudonyms
before they are used.

With pseudonym management, there is a trade-off between allowing for
pseudonym resynchronization (thanks to a permanent identity) and
being vulnerable to active attacks (in which the attacker forges
messages simulating a pseudonym desynchronization).

Indeed, a protocol using time-varying pseudonyms may want to
anticipate "desynchronization" situations such as, for instance, when
the peer believes that its current pseudonym is "pseudo1@bigco.com"
whereas the server believes this peer will use the pseudonym
"pseudo2@bigco.com" (which is the pseudonym the server has sent to
update "pseudo1@bigco.com").

Because pseudonym management adds complexity to the protocol and
implies this unsatisfactory trade-off, it was decided not to include
this feature in EAP-PSK.

However, EAP-PSK may trivially provide some protection when the
concern is to avoid the "real-life" identity of the user being
"discovered". For instance, let us take the example of user John Doe
that roams and connects to a Hot-Spot owned and operated by Wireless
Internet Service Provider (WISP) BAD. Suppose this user
authenticates to his home WISP (WISP GOOD) with an EAP method under
an identity (e.g., "john.doe@wispgood.com") that allows WISP BAD (or
an attacker) to recover his "real-life" identity, i.e., John Doe. An
example drawback of this is when a competitor of John Doe's WISP
wants to win John Doe as a new customer by sending him some special
targeted advertisement.

EAP-PSK can very simply thwart this attack, merely by avoiding to
provide John Doe with an NAI that allows easy recovery of his real-
life identity. It is believed that when an NAI that is not
correlated to a real-life identity is used, no valuable information
leaks because of the EAP method.

Indeed, the identity of the WISP used by a peer has to be disclosed
anyway in the realm portion of its NAI to allow AAA routing.
Moreover, the Medium Access Control Address of the peer's Network
Interface Card can generally be used to track the peer as efficiently
as a fixed NAI.

It is worth noting that the server systematically discloses its
identity, which may allow probing attacks. This may not be a problem
as the identity of the server is not supposed to remain secret. On
the contrary, users tend to want to know to whom they will be talking
in order to choose the right network to attach to.

8.15. Protected Ciphersuite Negotiation

EAP-PSK does not allow negotiating ciphersuites. Hence, it is not
vulnerable to negotiation attacks and does not implement protected
ciphersuite negotiation.

8.16. Confidentiality

Although EAP-PSK provides confidentiality in its protected channel,
it cannot claim to do so as per Section 7.2.1 of [3]: "A method
making this claim must support identity protection".

8.17. Cryptographic Binding

Since EAP-PSK is not intended to be tunneled within another protocol
that omits peer authentication, it does not implement cryptographic
binding.

8.18. Implementation of EAP-PSK

To really provide security, not only must a protocol be well thought-
out and correctly specified, but its implementation must take special
care.

For instance, implementing cryptographic algorithms requires special
skills since cryptographic software is vulnerable not only to
classical attacks (e.g., buffer overflow or missing checks) but also
to some special cryptographic attacks (e.g., side channels attacks
like timing ones; see [36]). In particular, care must be taken to
avoid such attacks in EAX implementation; please refer to [4] for a
note on this point.

An EAP-PSK implementation should use a good source of randomness to
generate the random numbers required in the protocol. Please refer
to [20] for more information on generating random numbers for
security applications.

Handling sensitive material (namely, keying material such as the PSK,
AK, KDK, etc.) should be done in a secure way (see, for instance,
[19] for guidance on secure deletion).

The specification of a repository for the PSK that EAP-PSK uses is
outside the scope of this document. In particular, nothing prevents
one from storing this PSK on a tamper-resistant device such as a
smart card rather than having it memorized or written down on a sheet
of paper. The choice of the PSK repository may have important
security impacts.

9. Security Claims

   This section provides the security claims required by [3].

   [a] Mechanism. EAP-PSK is based on symmetric cryptography (AES-128)
        and uses a 16-byte Pre-Shared Key (PSK).

   [b] Security claims. EAP-PSK provides:

        * Mutual authentication (see Section 8.1)

        * Integrity protection (see Section 8.3)

        * Replay protection (see Section 8.4)

        * Key derivation (see Section 8.7)

        * Dictionary attack resistance (see Section 8.6)

        * Session independence (see Section 8.9)

   [c] Key strength. EAP-PSK provides a 16-byte effective key
        strength.

   [d] Description of key hierarchy. Please see Section 2.1.

   [e] Indication of vulnerabilities. EAP-PSK does not provide:

        * Identity protection (see Section 8.14)

        * Confidentiality (see Section 8.16)

        * Fast reconnect (see Section 8.13)

        * Fragmentation (see Section 8.11)

        * Cryptographic binding (see Section 8.17)

        * Protected ciphersuite negotiation (see Section 8.15)

        * Perfect Forward Secrecy (see Section 8.10)

        * Key agreement: the session key is chosen by the peer (see
           Section 8.7)

        * Channel binding (see Section 8.12)

10. Acknowledgments

   This EAP method has been inspired by EAP-Archie and EAP-SIM. Many
   thanks to their respective authors: Jesse Walker (extra thanks to
   Jesse Walker for his thorough and challenging expert review of EAP-
   PSK), Russ Housley, Henry Haverinen, and Joseph Salowey.

   Thanks to

   o Henri Gilbert for some interesting discussions on the
      cryptographic parts of EAP-PSK.

   o Aurelien Magniez for his valuable feedback on network aspects of
      EAP-PSK, his curiosity and rigor that led to numerous
      improvements, and his help in the first implementation of EAP-PSK
      under Microsoft Windows and Freeradius.

   o Thomas Otto for his valuable feedback on EAP-PSK and the
      implementation of the first version of EAP-PSK under Xsupplicant.

   o Nancy Cam-Winget for some exchanges on EAP-PSK.

   o Jari Arkko and Bernard Aboba, the beloved EAP WG chairs, for the
      work they stimulate.

   Finally, thanks to Vir Z., who has brought a permanent and
   outstanding though discreet contribution to this protocol.

11. References

11.1. Normative References

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

   [2]   Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network
          Access Identifier", RFC 4282, December 2005.

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

   [4]   Bellare, M., Rogaway, P., and D. Wagner, "The EAX mode of
          operation", FSE 04, Springer-Verlag LNCS 3017, 2004.

   [5]   Gilbert, H., "The Security of One-Block-to-Many Modes of
          Operation", FSE 03, Springer-Verlag LNCS 2287, 2003.

   [6]   Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA
          Considerations Section in RFCs", BCP 26, RFC 2434,
          October 1998.

   [7]   National Institute of Standards and Technology, "Specification
          for the Advanced Encryption Standard (AES)", Federal
          Information Processing Standards (FIPS) 197, November 2001.

   [8]   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.

11.2. Informative References

   [9]   Aboba, B., Simon, D., Eronen, P., and H. Levkowetz,"Extensible
          Authentication Protocol (EAP) Key Management Framework", Work
          in Progress, October 2006.

   [10]   Aboba, B., Calhoun, P., Glass, S., Hiller, T., McCann, P.,
          Shiino, H., Zorn, G., Dommety, G., Perkins, C., Patil, B.,
          Mitton, D., Manning, S., Beadles, M., Walsh, P., Chen, X.,
          Sivalingham, S., Hameed, A., Munson, M., Jacobs, S., Lim, B.,
          Hirschman, B., Hsu, R., Xu, Y., Campell, E., Baba, S., and E.
          Jaques, "Criteria for Evaluating AAA Protocols for work
          Access", RFC 2989, November 2000.

   [11]   Aboba, B. and D. Simon, "PPP EAP TLS Authentication Protocol",
          RFC 2716, October 1999.

   [12]   Arkko, J. and H. Haverinen, "Extensible Authentication Protocol
          Method for 3rd Generation Authentication and Key Agreement
          (EAP-AKA)", RFC 4187, January 2006.

   [13]   Arkko, J. and P. Eronen, "Authenticated Service Information for
          the Extensible Authentication Protocol (EAP)", Work in
          Progress, October 2005.

   [14]   Bellare, M. and P. Rogaway, "Entity Authentication and Key
          Distribution", CRYPTO 93, Springer-Verlag LNCS 773, 1994.

   [15] Bellare, M., Pointcheval, D., and P. Rogaway, "Authenticated
         Key Exchange Secure Against Dictionary attacks", EUROCRYPT 00,
         Springer-Verlag LNCS 1807, 2000.

   [16] Bersani, F., "EAP shared key methods: a tentative synthesis of
         those proposed so far", Work in Progress, April 2004.

   [17] Bradner, S., "The Internet Standards Process -- Revision 3",
         BCP 9, RFC 2026, October 1996.

   [18] Carlson, J., Aboba, B., and H. Haverinen, "EAP SRP-SHA1
         Authentication Protocol", Work in Progress, July 2001.

   [19] Department of Defense of the United States, "National
         Industrial Security Program Operating Manual", DoD 5220-22M,
         January 1995.

   [20] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
         Requirements for Security", BCP 106, RFC 4086, June 2005.

   [21] Funk, P. and S. Blake-Wilson, "EAP Tunneled TLS Authentication
         Protocol (EAP-TTLS)", Work in Progress, July 2004.

   [22] Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time
         Password System", RFC 2289, February 1998.

   [23] Halpern, J. and Y. Moses, "Knowledge and common knowledge in a
         distributed environment", Journal of the ACM 37:3, 1990.

   [24] Haverinen, H. and J. Salowey, "Extensible Authentication
         Protocol Method for Global System for Mobile Communications
         (GSM) Subscriber Identity Modules (EAP-SIM)", RFC 4186,
         January 2006.

   [25] Huitema, C., Postel, J., and S. Crocker, "Not All RFCs are
         Standards", RFC 1796, April 1995.

   [26] Institute of Electrical and Electronics Engineers, "Local and
         Metropolitan Area Networks: Port-Based Network Access Control",
         IEEE Standard 802.1X, September 2001.

   [27] Institute of Electrical and Electronics Engineers, "Approved
         Draft Supplement to Standard for Telecommunications and
         Information Exchange Between Systems-LAN/MAN Specific
         Requirements – Part 11: Wireless LAN Medium Access Control
         (MAC) and Physical Layer (PHY) Specifications: Specification
         for Enhanced Security", IEEE 802.11i-2004, 2004.

[28] Institute of Electrical and Electronics Engineers, "Standard
      for Telecommunications and Information Exchange Between Systems
      - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium
      Access Control (MAC) and Physical Layer (PHY) Specifications",
      IEEE Standard 802.11, 1999.

[29] Iwata, T. and K. Kurosawa, "OMAC: One-Key CBC MAC", FSE 03,
      Springer-Verlag LNCS 2887, 2003.

[30] Jablon, D., "The SPEKE Password-Based Key Agreement Methods",
      Work in Progress, November 2002.

[31] Josefsson, S., "The EAP SecurID(r) Mechanism", Work in
      Progress, February 2002.

[32] Josefsson, S., Palekar, A., Simon, D., and G. Zorn, "Protected
      EAP Protocol (PEAP) Version 2", Work in Progress, October 2004.

[33] Kaliski, B., "PKCS #5: Password-Based Cryptography
      Specification Version 2.0", RFC 2898, September 2000.

[34] Kamath, V. and A. Palekar, "Microsoft EAP CHAP Extensions",
      Work in Progress, April 2004.

[35] Kent, S., "IP Authentication Header", RFC 4302, December 2005

[36] Kocher, P., "Timing Attacks on Implementations of Diffie-
      Hellman, RSA, DSS, and Other Systems", CRYPTO 96, Springer-
      Verlag LNCS 1109, 1996.

[37] Krawczyk, H., "SIGMA: the 'SIGn-and-MAc' Approach to
      Authenticated Diffie-Hellman and its Use in the IKE Protocols",
      CRYPTO 03, Springer-Verlag LNCS 2729, June 2003.

[38] MacNally, C., "Cisco LEAP protocol description",
      September 2001, available from
      <http://www.missl.cs.umd.edu/wireless/ethereal/leap.txt>.

[39] Metz, C., "OTP Extended Responses", RFC 2243, November 1997.

[40] Menezes, A., van Oorschot, P., and S. Vanstone, "Handbook of
      Applied Cryptography", CRC Press , 1996.

[41] National Institute of Standards and Technology, "Password
      Usage", Federal Information Processing Standards (FIPS) 112,
      May 1985.

   [42]   Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "The
          Flexible Authentication via Secure Tunneling Extensible
          Authentication Protocol Method (EAP-FAST)", Work in Progress,
          October 2006.

   [43]   Schneier, B., Mudge, and D. Wagner, "Cryptanalysis of
          Microsoft's PPTP Authentication Extensions (MS-CHAPv2)",
          CQRE 99, Springer-Verlag LNCS 1740, October 1999.

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

   [45]   Simpson, W., "PPP Challenge Handshake Authentication Protocol
          (CHAP)", RFC 1994, August 1996.

   [46]   Tschofenig, H., Kroeselberg, D., Pashalidis, A., Ohba, Y., and
          F. Bersani, "EAP IKEv2 Method", Work in Progress, October 2006.

   [47]   Walker, J. and R. Housley, "The EAP Archie Protocol", Work in
          Progress, June 2003.

   [48]   Wi-Fi Alliance, "Wi-Fi Protected Access, version 2.0",
          April 2003.

   [49]   Wright, J., "Weaknesses in LEAP Challenge/Response", Defcon 03,
          August 2003.

   [50]   Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for
          Transport Layer Security (TLS)", RFC 4279, December 2005.

Appendix A. Generation of the PSK from a Password - Discouraged

   It is formally discouraged to use a password to generate the PSK,
   since this opens the door to exhaustive search or dictionary attacks,
   two attacks that would not otherwise be possible.

   EAP-PSK only provides a 16-byte key strength when a 16-byte PSK is
   drawn at random from the set of all possible 16-byte strings.

   However, as people will probably do this anyway, guidance is provided
   hereafter to generate the PSK from a password.

   For some hints on how passwords should be selected, please refer to
   [41].

   The technique presented herein is drawn from [33]. It is intended to
   try to mitigate the risks associated with password usage in
   cryptography, typically dictionary attacks.

   If the binary representation of the password is strictly fewer than
   16 bytes long (which by the way means that the chosen password is
   probably weak because it is too short), then it is padded to 16 bytes
   with zeroes as its high-order bits.

   If the binary representation of the password is strictly more than 16
   bytes long, then it is hashed down to exactly 16 bytes using the
   Matyas-Meyer-Oseas hash (please refer to [40] for a description of
   this hash. Using the notation of Figure 9.3 of [40], g is the
   identity function and E is AES-128 in our construction.) with
   IV=0x0123456789ABCDEFFEDCBA9876543210 (this value has been
   arbitrarily selected).

   We now assume that we have a 16-byte number derived from the initial
   password (that can be the password itself if its binary
   representation is exactly 16 bytes long). We shall call this number
   P16.

   Following the notations used in [33], the PSK is derived thanks to
   PBKDF2 instantiated with:

   o P16 as P

   o The first 96 bits of the XOR of the peer and server NAIs as Salt
      (zero-padded in the high-order bits if necessary).

   o 5000 as c

   o 16 as dkLen

Although this gives better protection than nothing, this derivation does not stricto sensu protect against dictionary attacks. It only makes dictionary precomputation harder.

Authors' Addresses

    Florent Bersani
    France Telecom R&D
    38, rue du General Leclerc
    Issy-Les-Moulineaux 92794 Cedex 9
    FR

    EMail: bersani_florent@yahoo.fr

    Hannes Tschofenig
    Siemens Networks GmbH & Co KG
    Otto-Hahn-Ring 6
    Munich 81739
    GE

    EMail: Hannes.Tschofenig@siemens.com

Full Copyright Statement

Intellectual Property

Acknowledgement