              Extensible Authentication Protocol (EAP) Authentication
                           Using Only a Password

Abstract

   This memo describes an Extensible Authentication Protocol (EAP)
   method, EAP-pwd, which uses a shared password for authentication.
   The password may be a low-entropy one and may be drawn from some set
   of possible passwords, like a dictionary, which is available to an
   attacker. The underlying key exchange is resistant to active attack,
   passive attack, and dictionary attack.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Background

   The predominant access method for the Internet today is that of a
   human using a username and password to authenticate to a computer
   enforcing access control. Proof of knowledge of the password
   authenticates the human and computer.

   Typically these passwords are not stored on a user's computer for
   security reasons and must be entered each time the human desires
   network access. Therefore, the passwords must be ones that can be
   repeatedly entered by a human with a low probability of error. They
   will likely not possess high-entropy, and it may be assumed that an
   adversary with access to a dictionary will have the ability to guess
   a user's password. It is therefore desirable to have a robust
   authentication method that is secure even when used with a weak
   password in the presence of a strong adversary.

   EAP-pwd is an EAP method that addresses the problem of password-based
   authenticated key exchange -- using a possibly weak password for
   authentication to derive an authenticated and cryptographically
   strong shared secret. This problem was first described by Bellovin
   and Merritt in [BM92] and [BM93]. There have been a number of
   subsequent suggestions ([JAB96], [LUC97], [BMP00], and others) for
   password-based authenticated key exchanges.

1.2. Keyword Definitions

   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 RFC 2119 [RFC2119].

1.3. Requirements

   Any protocol that claims to solve the problem of password-
   authenticated key exchange must be resistant to active, passive, and
   dictionary attack and have the quality of forward secrecy. These
   characteristics are discussed further in the following sections.

1.3.1. Resistance to Passive Attack

   A passive, or benign, attacker is one that merely relays messages
   back and forth between the peer and server, faithfully, and without
   modification. The contents of the messages are available for
   inspection, but that is all. To achieve resistance to passive
   attack, such an attacker must not be able to obtain any information
   about the password or anything about the resulting shared secret from

watching repeated runs of the protocol. Even if a passive attacker
is able to learn the password, she will not be able to determine any
information about the resulting secret shared by the peer and server.

1.3.2. Resistance to Active Attack

An active attacker is able to modify, add, delete, and replay
messages sent between protocol participants. For this protocol to be
resistant to active attack, the attacker must not be able to obtain
any information about the password or the shared secret by using any
of its capabilities. In addition, the attacker must not be able to
fool a protocol participant into thinking that the protocol completed
successfully.

It is always possible for an active attacker to deny delivery of a
message critical in completing the exchange. This is no different
than dropping all messages and is not an attack against the protocol.

1.3.3. Resistance to Dictionary Attack

For this protocol to be resistant to dictionary attack, any advantage
an adversary can gain must be directly related to the number of
interactions she makes with an honest protocol participant and not
through computation. The adversary will not be able to obtain any
information about the password except whether a single guess from a
single protocol run is correct or incorrect.

1.3.4. Forward Secrecy

Compromise of the password must not provide any information about the
secrets generated by earlier runs of the protocol.

2. Specification of EAP-pwd

2.1. Notation

The following notation is used in this memo:

peer-ID
    The peer's identity, the peer NAI [RFC4282].

server-ID
    A string that identifies the server to the peer.

password
    The password shared between the peer and server.

y = H(x)
     The binary string x is given to a function H, which produces a
     fixed-length output y.

a | b
     The concatenation of string a with string b.

[a]b
     A string consisting of the single bit "a" repeated "b" times.

x mod y
     The remainder of division of x by y. The result will be between
     0 and y.

g^x mod p
     The multiplication of the value "g" with itself "x" times, modulo
     the value "p".

inv(Q)
     The inverse of an element, Q, from a finite field.

len(x)
     The length in bits of the string x.

chop(x, y)
     The reduction of string x, being at least y bits in length, to y
     bits.

PRF(x,y)
     A pseudo-random function that takes a key, x, and variable-length
     data, y, and produces a fixed-length output that cannot be
     distinguished (with a significant advantage) from a random
     source.

LSB(x)
     Returns the least-significant bit of the bitstring "x".

Ciphersuite
     An encoding of a group to use with EAP-pwd, the definition of
     function H, and a PRF, in that order.

MK
     The Master Key is generated by EAP-pwd. This is a high-entropy
     secret whose length depends on the random function used.

MSK
>   The Master Session Key exported by EAP-pwd. This is a high-
>   entropy secret 512 bits in length.

EMSK
>   The Extended Master Session Key exported by EAP-pwd. This is a
>   high-entropy secret 512 bits in length.

2.2. Discrete Logarithm Cryptography

This protocol uses discrete logarithm cryptography to achieve
authentication and key agreement (see [SP800-56A]). Each party to
the exchange derives ephemeral keys with respect to a particular set
of domain parameters (referred to here as a "group"). A group can be
based on Finite Field Cryptography (FFC) or Elliptic Curve
Cryptography (ECC).

2.2.1. Finite Field Cryptography

Domain parameters for the FFC groups used by EAP-pwd include:

o A prime, p, determining a prime field GF(p), the integers modulo
   p. The FFC group will be a subgroup of GF(p)*, the multiplicative
   group of non-zero elements in GF(p). The group operation for FFC
   groups is multiplication modulo p.

o An element, G, in GF(p)* which serves as a generator for the FFC
   group. G is chosen such that its multiplicative order is a
   sufficiently large prime divisor of ((p-1)/2).

o A prime, r, which is the multiplicative order of G, and thus also
   the size of the cryptographic subgroup of GF(p)* that is generated
   by G.

An integer scalar, x, acts on an FFC group element, Y, via
exponentiation modulo p -- Y^x mod p.

The inverse function for an FFC group is defined such that the
product of an element and its inverse modulo the group prime equals
one (1). In other words,

    (q * inv(q)) mod p = 1

EAP-pwd uses an IANA registry for the definition of groups. Some FFC
groups in this registry are based on safe primes and the order is not
included in the domain parameters. In this case only, the order, r,
MUST be computed as the prime minus one divided by two -- (p-1)/2.
If the definition of the group includes an order in its domain

parameters, that value MUST be used in this exchange when an order is
called for. If an FFC group definition does not have an order in its
domain parameters and it is not based on a safe prime, it MUST NOT be
used with EAP-pwd.

2.2.2. Elliptic Curve Cryptography

Domain parameters for the ECC groups used by EAP-pwd include:

o A prime, p, determining a prime field GF(p). The cryptographic
   group will be a subgroup of the full elliptic curve group that
   consists of points on an elliptic curve -- elements from GF(p)
   that satisfy the curve's equation -- together with the "point at
   infinity" that serves as the identity element. The group
   operation for ECC groups is addition of points on the elliptic
   curve.

o Elements a and b from GF(p) that define the curve's equation. The
   point (x, y) in GF(p) x GF(p) is on the elliptic curve if and only
   if (y^2 - x^3 - a*x - b) mod p equals zero (0).

o A point, G, on the elliptic curve, which serves as a generator for
   the ECC group. G is chosen such that its order, with respect to
   elliptic curve addition, is a sufficiently large prime.

o A prime, r, which is the order of G, and thus is also the size of
   the cryptographic subgroup that is generated by G.

o A co-factor, f, defined by the requirement that the size of the
   full elliptic curve group (including the "point at infinity") is
   the product of f and r.

An integer scalar, x, acts on an ECC group element, Y, via repetitive
addition (Y is added to itself x times), also called point
multiplication -- x * Y.

The inverse function for an ECC group is defined such that the sum of
an element and its inverse is the "point at infinity" (the identity
for elliptic curve point addition). In other words,

    Q + inv(Q) = "O"

Only ECC groups over GF(p) can be used by EAP-pwd. ECC groups over
GF(2^m) SHALL NOT be used by EAP-pwd. While such groups exist in the
IANA registry used by EAP-pwd, their use in EAP-pwd is not defined.
In addition, ECC groups with a co-factor greater than one (1) SHALL
NOT be used by EAP-pwd. At the time of publication, no such groups
existed in the IANA registry used by EAP-pwd.

2.3. Assumptions

   In order to see how the protocol addresses the requirements above
   (see Section 1.3), it is necessary to state some assumptions under
   which the protocol can be evaluated. They are:

   1. Function H maps a binary string of indeterminate length onto a
       fixed binary string that is x bits in length.

           H: {0,1}^* --> {0,1}^x

   2. Function H is a "random oracle" (see [RANDOR]). Given knowledge
       of the input to H, an adversary is unable to distinguish the
       output of H from a random data source.

   3. Function H is a one-way function. Given the output of H, it is
       computationally infeasible for an adversary to determine the
       input.

   4. For any given input to function H, each of the 2^x possible
       outputs are equally probable.

   5. The discrete logarithm problem for the chosen group is hard.
       That is, given g, p, and $y = g^x \bmod p$, it is computationally
       infeasible to determine x. Similarly, for an ECC group given the
       curve definition, a generator G, and $Y = x * G$, it is
       computationally infeasible to determine x.

   6. There exists a pool of passwords from which the password shared
       by the peer and server is drawn. This pool can consist of words
       from a dictionary, for example. Each password in this pool has
       an equal probability of being the shared password. All potential
       attackers have access to this pool of passwords.

2.4. Instantiating the Random Function

   The protocol described in this memo uses a random function, H. As
   noted in Section 2.3, this is a "random oracle" as defined in
   [RANDOR]. At first glance, one may view this as a hash function. As
   noted in [RANDOR], though, hash functions are too structured to be
   used directly as a random oracle. But they can be used to
   instantiate the random oracle.

   The random function, H, in this memo is instantiated by HMAC-SHA256
   (see [RFC4634]) with a key whose length is 32 octets and whose value
   is zero. In other words,

       H(x) = HMAC-SHA-256([0]32, x)

2.5. Key Derivation Function

   The keys output by this protocol, MSK and EMSK, are each 512 bits in
   length. The shared secret that results from the successful
   termination of this protocol is only 256 bits. Therefore, it is
   necessary to stretch the shared secret using a key derivation
   function (KDF).

   The KDF used in this protocol has a counter-mode with feedback
   construction using a generic pseudo-random function (PRF), according
   to [SP800-108]. The specific value of the PRF is specified along
   with the random function and group when the server sends the first
   EAP-pwd packet to the peer.

   The KDF takes a key to stretch, a label to bind into the key, and an
   indication of the desired length of the output in bits. It uses two
   internal variables, i and L, each of which is 16 bits in length and
   is represented in network order. Algorithmically, it is:

```
            KDF(key, label, length) {
              i = 1
              L = length
              K(1) = PRF(key, i | label | L)
              res = K(1)
              while (len(res) < length)
              do
                i = i + 1
                K(i) = PRF(key, K(i-1) | i | label | L)
                res = res | K(i)
              done
              return chop(res, length)
            }
```

                  Figure 1: Key Derivation Function

2.6. Random Numbers

   The security of EAP-pwd relies upon each side, the peer and server,
   producing quality secret random numbers. A poor random number chosen
   by either side in a single exchange can compromise the shared secret
   from that exchange and open up the possibility of dictionary attack.

   Producing quality random numbers without specialized hardware entails
   using a cryptographic mixing function (like a strong hash function)
   to distill entropy from multiple, uncorrelated sources of information
   and events. A very good discussion of this can be found in
   [RFC4086].

2.7. Representation and Processing of Input Strings

2.7.1. Identity Strings

   The strings representing the server identity and peer identity MUST
   follow the requirements of [RFC4282] for Network Access Identifiers.
   This ensures a canonical representation of identities by both ends of
   the conversation prior to their use in EAP-pwd.

2.7.2. Passwords

   EAP-pwd requires passwords be input as binary strings. For the
   protocol to successfully terminate, each side must produce identical
   binary strings from the password. This imposes processing
   requirements on a password prior to its use.

   Three techniques for password pre-processing exist for EAP-pwd:

   o   None: The input password string SHALL be treated as an ASCII
        string or a hexadecimal string with no treatment or normalization
        performed. The output SHALL be the binary representation of the
        input string.

   o   RFC 2759: The input password string SHALL be processed to produce
        the output PasswordHashHash, as defined in [RFC2759], including
        any approved errata to [RFC2759]. This technique is useful when
        the server does not have access to the plaintext password.

   o   SASLprep: The input password string is processed according to the
        rules of the [RFC4013] profile of [RFC3454]. A password SHALL be
        considered a "stored string" per [RFC3454], and unassigned code
        points are therefore prohibited. The output SHALL be the binary
        representation of the processed UTF-8 character string.
        Prohibited output and unassigned codepoints encountered in
        SASLprep pre-processing SHALL cause a failure of pre-processing,
        and the output SHALL NOT be used with EAP-pwd.

   Changing a password is out of scope of EAP-pwd, but due to the
   ambiguities in the way internationalized character strings are
   handled, 1) it SHOULD be done using SASLprep to ensure a canonical
   representation of the new password is stored on the server, and 2)
   subsequent invocations of EAP-pwd SHOULD use SASLprep to ensure that
   the client generates an identical binary string from the input
   password.

2.8. Protocol

2.8.1. Overview

   EAP is a two-party protocol spoken between an EAP peer and an
   authenticator. For scaling purposes, the functionality of the
   authenticator that speaks EAP is frequently broken out into a stand-
   alone EAP server. In this case, the EAP peer communicates with an
   EAP server through the authenticator, with the authenticator merely
   being a passthrough.

   An EAP method defines the specific authentication protocol being used
   by EAP. This memo defines a particular method and therefore defines
   the messages sent between the EAP server (or the "EAP server"
   functionality in an authenticator if it is not broken out) and the
   EAP peer for the purposes of authentication and key derivation.

2.8.2. Message Flows

   EAP-pwd defines three message exchanges: an Identity exchange, a
   Commit exchange, and a Confirm exchange. A successful authentication
   is shown in Figure 2.

   The peer and server use the Identity exchange to discover each
   other's identities and to agree upon a Ciphersuite to use in the
   subsequent exchanges; in addition, the EAP Server uses the EAP-pwd-
   ID/Request message to inform the client of any password pre-
   processing that may be required. In the Commit exchange, the peer
   and server exchange information to generate a shared key and also to
   bind each other to a particular guess of the password. In the
   Confirm exchange, the peer and server prove liveness and knowledge of
   the password by generating and verifying verification data.

```
                +--------+    EAP-pwd-ID/Request    +--------+
                |  EAP   |<-----------------------------------| EAP    |
                |  peer  |                                    | server |
                |        |  EAP-pwd-ID/Response               |        |
                |        |----------------------------------->|        |
                |        |                                    |        |
                |        |        EAP-pwd-Commit/Request       |        |
                |        |<-----------------------------------|        |
                |        |                                    |        |
                |        |  EAP-pwd-Commit/Response            |        |
                |        |----------------------------------->|        |
                |        |                                    |        |
                |        |        EAP-pwd-Confirm/Request      |        |
                |        |<-----------------------------------|        |
                |        |                                    |        |
                |        |  EAP-pwd-Confirm/Response           |        |
                |        |----------------------------------->|        |
                |        |                                    |        |
                |        |            EAP-Success              |        |
                |        |<-----------------------------------|        |
                +--------+                                    +--------+
```
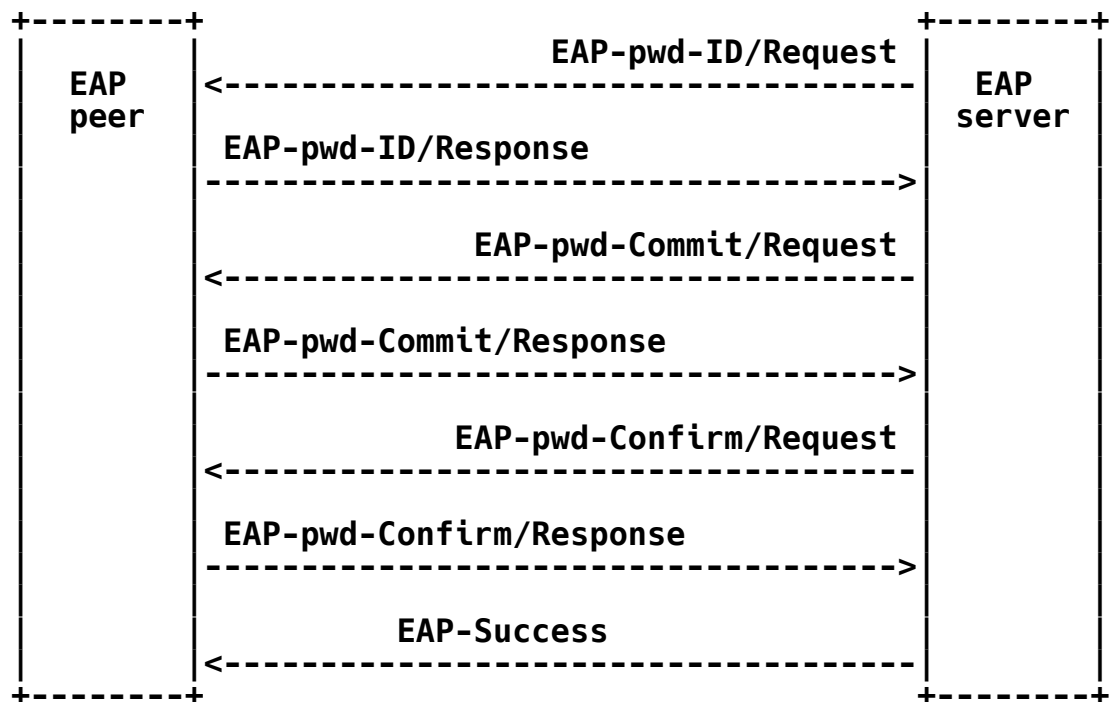
                  Figure 2: A Successful EAP-pwd Exchange

   The components of the EAP-pwd-* messages are as follows:

   EAP-pwd-ID/Request
        Ciphersuite, Token, Password Processing Method, Server_ID

   EAP-pwd-ID/Response
        Ciphersuite, Token, Password Processing Method, Peer_ID

   EAP-pwd-Commit/Request
        Scalar_S, Element_S

   EAP-pwd-Commit/Response
        Scalar_P, Element_P

   EAP-pwd-Confirm/Request
        Confirm_S

   EAP-pwd-Confirm/Response
        Confirm_P

2.8.3. Fixing the Password Element

   Once the EAP-pwd-ID exchange is completed, the peer and server use
   each other's identities and the agreed upon ciphersuite to fix an
   element in the negotiated group called the Password Element (PWE or
   pwe, for an element in an ECC group or an FFC group, respectively).
   The resulting element must be selected in a deterministic fashion
   using the password but must result in selection of an element that
   will not leak any information about the password to an attacker.
   From the point of view of an attacker who does not know the password,
   the Password Element will be a random element in the negotiated
   group.

   To properly fix the Password Element, both parties must have a common
   view of the string "password". Therefore, if a password pre-
   processing algorithm was negotiated during the EAP-pwd-ID exchange,
   the client MUST perform the specified password pre-processing prior
   to fixing the Password Element.

   Fixing the Password Element involves an iterative hunting-and-pecking
   technique using the prime from the negotiated group's domain
   parameter set and an ECC- or FFC-specific operation depending on the
   negotiated group.

   First, an 8-bit counter is set to the value one (1). Then, the
   agreed-upon random function is used to generate a password seed from
   the identities and the anti-clogging token from the EAP-pwd-ID
   exchange (see Section 2.8.5.1):

      pwd-seed = H(token | peer-ID | server-ID | password | counter)

   Then, the pwd-seed is expanded using the KDF from the agreed-upon
   Ciphersuite out to the length of the prime:

      pwd-value = KDF(pwd-seed, "EAP-pwd Hunting And Pecking", len(p))

   If the pwd-value is greater than or equal to the prime, p, the
   counter is incremented, and a new pwd-seed is generated and the
   hunting-and-pecking continues. If pwd-value is less than the prime,
   p, it is passed to the group-specific operation which either returns
   the selected Password Element or fails. If the group-specific
   operation fails, the counter is incremented, a new pwd-seed is
   generated, and the hunting-and-pecking continues. This process
   continues until the group-specific operation returns the Password
   Element.

2.8.3.1. ECC Operation for PWE

   The group-specific operation for ECC groups uses pwd-value, pwd-seed,
   and the equation for the curve to produce the Password Element.
   First, pwd-value is used directly as the x-coordinate, x, with the
   equation for the elliptic curve, with parameters a and b from the
   domain parameter set of the curve, to solve for a y-coordinate, y.
   If there is no solution to the quadratic equation, this operation
   fails and the hunting-and-pecking process continues. If a solution
   is found, then an ambiguity exists as there are technically two
   solutions to the equation and pwd-seed is used to unambiguously
   select one of them. If the low-order bit of pwd-seed is equal to the
   low-order bit of y, then a candidate PWE is defined as the point
   (x, y); if the low-order bit of pwd-seed differs from the low-order
   bit of y, then a candidate PWE is defined as the point (x, p - y),
   where p is the prime over which the curve is defined. The candidate
   PWE becomes PWE, and the hunting and pecking terminates successfully.

   Algorithmically, the process looks like this:

```
        found = 0
        counter = 1
        do {
          pwd-seed = H(token | peer-ID | server-ID | password | counter)
          pwd-value = KDF(pwd-seed, "EAP-pwd Hunting And Pecking", len(p))
          if (pwd-value < p)
          then
            x = pwd-value
            if ( (y = sqrt(x^3 + ax + b)) != FAIL)
            then
              if (LSB(y) == LSB(pwd-seed))
              then
                PWE = (x, y)
              else
                PWE = (x, p-y)
              fi
              found = 1
          fi
        fi
        counter = counter + 1
      } while (found == 0)
```

                       Figure 3: Fixing PWE for ECC Groups

2.8.3.2. FFC Operation for pwe

   The group-specific operation for FFC groups takes pwd-value, and the
   prime, p, and order, r, from the group's domain parameter set (see
   Section 2.2.1 when the order is not part of the defined domain
   parameter set) to directly produce a candidate Password Element, pwe,
   by exponentiating the pwd-value to the value ((p-1)/r) modulo the
   prime. If the result is greater than one (1), the candidate pwe
   becomes pwe, and the hunting and pecking terminates successfully.

   Algorithmically, the process looks like this:

```
      found = 0
      counter = 1
      do {
        pwd-seed = H(token | peer-ID | server-ID | password | counter)
        pwd-value = KDF(pwd-seed, "EAP-pwd Hunting And Pecking", len(p))
        if (pwd-value < p)
        then
          pwe = pwd-value ^ ((p-1)/r) mod p
          if (pwe > 1)
          then
            found = 1
          fi
        fi
        counter = counter + 1
      } while (found == 0)
```

                     Figure 4: Fixing PWE for FFC Groups

2.8.4. Message Construction

   After the EAP-pwd Identity exchange, the construction of the
   components of subsequent messages depends on the type of group from
   the ciphersuite (ECC or FFC). This section provides an overview of
   the authenticated key exchange. For a complete description of
   message generation and processing, see Sections 2.8.5.2 and 2.8.5.3.

2.8.4.1. ECC Groups

   Using the mapping function F() defined in Section 2.2.2 and the group
   order r:

   Server: EAP-pwd-Commit/Request
      - choose two random numbers, 1 < s_rand, s_mask < r
      - compute Scalar_S = (s_rand + s_mask) mod r
      - compute Element_S = inv(s_mask * PWE)

Element_S and Scalar_S are used to construct EAP-pwd-Commit/Request

Peer: EAP-pwd-Commit/Response
   - choose two random numbers, 1 < p_rand, p_mask < r
   - compute Scalar_P = (p_rand + p_mask) mod r
   - compute Element_P = inv(p_mask * PWE)

Element_P and Scalar_P are used to construct EAP-pwd-Commit/Response

Server: EAP-pwd-Confirm/Request
   - compute KS = (s_rand * (Scalar_P * PWE + Element_P))
   - compute ks = F(KS)
   - compute Confirm_S = H(ks | Element_S | Scalar_S |
                          Element_P | Scalar_P | Ciphersuite)

Confirm_S is used to construct EAP-pwd-Confirm/Request

Peer: EAP-pwd-Confirm/Response
   - compute KP = (p_rand * (Scalar_S * PWE + Element_S)),
   - compute kp = F(KP)
   - compute Confirm_P = H(kp | Element_P | Scalar_P |
                          Element_S | Scalar_S | Ciphersuite)

Confirm_P is used to construct EAP-pwd-Confirm/Response

The EAP Server computes the shared secret as:
  MK = H(ks | Confirm_P | Confirm_S)

The EAP Peer computes the shared secret as:
  MK = H(kp | Confirm_P | Confirm_S)

The MSK and EMSK are derived from MK per Section 2.9.

2.8.4.2. FFC Groups

There is no mapping function, F(), required for an FFC group. Using
the order, r, for the group (see Section 2.2.1 when the order is not
part of the defined domain parameters):

Server: EAP-pwd-Commit/Request
   - choose two random numbers, 1 < s_rand, s_mask < r
   - compute Scalar_S = (s_rand + s_mask) mod r
   - compute Element_S = inv(pwe^s_mask mod p)

Element_S and Scalar_S are used to construct EAP-pwd-Commit/Request

```
   Peer: EAP-pwd-Commit/Response
      - choose random two numbers, 1 < p_rand, p_mask < r
      - compute Scalar_P = (p_rand + p_mask) mod r
      - compute Element_P = inv(pwe^p_mask mod p)

    Element_P and Scalar_P are used to construct EAP-pwd-Commit/Response

   Server: EAP-pwd-Confirm/Request
      - compute ks = ((pwe^Scalar_P mod p) * Element_P)^s_rand mod p
      - compute Confirm_S = H(ks | Element_S | Scalar_S |
                               Element_P | Scalar_P | Ciphersuite)

    Confirm_S is used to construct EAP-pwd-Confirm/Request

   Peer: EAP-pwd-Confirm/Response
      - compute kp = ((pwe^Scalar_S mod p) * Element_S)^p_rand mod p
      - compute Confirm_P = H(kp | Element_P | Scalar_P |
                               Element_S | Scalar_S | Ciphersuite)

    Confirm_P is used to construct EAP-pwd-Confirm/Request

   The EAP Server computes the shared secret as:
     MK = H(ks | Confirm_P | Confirm_S)

   The EAP Peer computes the shared secret as:
     MK = H(kp | Confirm_P | Confirm_S)
```

   The MSK and EMSK are derived from MK per Section 2.9.

2.8.5. Message Processing

2.8.5.1. EAP-pwd-ID Exchange

   Although EAP provides an Identity method to determine the identity of
   the peer, the value in the Identity Response may have been truncated
   or obfuscated to provide privacy or decorated for routing purposes
   [RFC3748], making it inappropriate for usage by the EAP-pwd method.
   Therefore, the EAP-pwd-ID exchange is defined for the purpose of
   exchanging identities between the peer and server.

   The EAP-pwd-ID/Request contains the following quantities:

   o a ciphersuite

   o a representation of the server's identity per Section 2.7.1

o   an anti-clogging token

o   a password pre-processing method

The ciphersuite specifies the finite cyclic group, random function,
and PRF selected by the server for use in the subsequent
authentication exchange.

The value of the anti-clogging token MUST be unpredictable and SHOULD
NOT be from a source of random entropy. The purpose of the anti-
clogging token is to provide the server an assurance that the peer
constructing the EAP-pwd-ID/Response is genuine and not part of a
flooding attack.

A password pre-processing method is communicated to ensure
interoperability by producing a canonical representation of the
password string between the peer and server (see Section 2.7.2).

The EAP-pwd-ID/Request is constructed according to Section 3.2.1 and
is transmitted to the peer.

Upon receipt of an EAP-pwd-ID/Request, the peer determines whether
the ciphersuite and pre-processing method are acceptable. If not,
the peer MUST respond with an EAP-NAK. If acceptable, the peer
responds to the EAP-pwd-ID/Request with an EAP-pwd-ID/Response,
constructed according to Section 3.2.1, that acknowledges the
Ciphersuite, token, and pre-processing method and then adds its
identity. After sending the EAP-pwd-ID/Response, the peer has the
identity of the server (from the Request), its own identity (it
encoded in the Response), a password pre-processing algorithm, and it
can compute the Password Element as specified in Section 2.8.3. The
Password Element is stored in state allocated for this exchange.

The EAP-pwd-ID/Response acknowledges the Ciphersuite from the
Request, acknowledges the anti-clogging token from the Request
providing a demonstration of "liveness" on the part of the peer, and
contains the identity of the peer. Upon receipt of the Response, the
server verifies that the Ciphersuite acknowledged by the peer is the
same as that sent in the Request and that the anti-clogging token
added by the peer in the Response is the same as that sent in the
Request. If Ciphersuites or anti-clogging tokens differ, the server
MUST respond with an EAP-Failure message. If the anti-clogging
tokens are the same, the server knows the peer is an active
participant in the exchange. If the Ciphersuites are the same, the
server now knows its own identity (it encoded in the Request) and the
peer's identity (from the Response) and can compute the Password

Element according to Section 2.8.3. The server stores the Password
Element in state it has allocated for this exchange. The server then
initiates an EAP-pwd-Commit exchange.

2.8.5.2. EAP-pwd-Commit Exchange

The server begins the EAP-pwd-Confirm exchange by choosing two random
numbers, s_rand and s_mask, between 1 and r (where r is described in
Section 2.1 according to the group established in Section 2.8.5.1)
such that their sum modulo r is greater than one (1). It then
computes Element_S and Scalar_S as defined in Section 2.8.4 and
constructs an EAP-pwd-Commit/Request according to Section 3.2.2.
Element_S and Scalar_S are added to the state allocated for this
exchange, and the EAP-pwd-Commit/Request is transmitted to the peer.

Upon receipt of the EAP-pwd-Commit/Request, the peer validates the
length of the entire payload based upon the expected lengths of
Element_S and Scalar_S (which are fixed according to the length of
the agreed-upon group). If the length is incorrect, the peer MUST
terminate the exchange. If the length is correct, Element_S and
Scalar_S are extracted from the EAP-pwd-Commit/Request. Scalar_S is
then checked to ensure it is between 1 and r, exclusive. If it is
not, the peer MUST terminate the exchange. If it is, Element_S MUST
be validated depending on the type of group -- Element validation for
FFC groups is described in Section 2.8.5.2.1, and Element validation
for ECC groups is described in Section 2.8.5.2.2. If validation is
successful, the peer chooses two random numbers, p_rand and p_mask,
between 1 and r (where r is described in Section 2.1 according to the
group established in Section 2.8.5.1) such that their sum modulo r is
greater than one (1), and computes Element_P and Scalar_P. Next, the
peer computes kp from p_rand, Element_S, Scalar_S, and the Password
Element according to Section 2.8.4. If kp is the "identity element"
-- the point at infinity for an ECC group or the value one (1) for an
FFC group -- the peer MUST terminate the exchange. If not, the peer
uses Element_P and Scalar_P to construct an EAP-pwd-Commit/Response
according to Section 3.2.2 and transmits the EAP-pwd-Commit/Response
to the server.

Upon receipt of the EAP-pwd-Commit/Response, the server validates the
length of the entire payload based upon the expected lengths of
Element_P and Scalar_P (which are fixed according to the agreed-upon
group). If the length is incorrect, the server MUST respond with an
EAP-Failure message, and it MUST terminate the exchange and free up
any state allocated. If the length is correct, Scalar_P and
Element_P are extracted from the EAP-pwd-Commit/Response and compared
to Scalar_S and Element_S. If Scalar_P equals Scalar_S and Element_P
equals Element_S, it indicates a reflection attack and the server
MUST respond with an EAP-failure and terminate the exchange. If they

differ, Scalar_P is checked to ensure it is between 1 and r,
exclusive. If not the server MUST respond with an EAP-failure and
terminate the exchange. If it is, Element_P is verified depending on
the type of group -- Element validation for FFC groups is described
in Section 2.8.5.2.1, and Element validation for ECC groups is
described in Section 2.8.5.2.2. If validation is successful, the
server computes ks from s_rand, Element_P, Scalar_P, and the Password
Element according to Section 2.8.4. If ks is the "identity element"
-- the point at infinity for an ECC group or the value one (1) for an
FFC group -- the server MUST respond with an EAP-failure and
terminate the exchange. Otherwise, the server initiates an EAP-pwd-
Confirm exchange.

2.8.5.2.1. Element Validation for FFC Groups

A received FFC Element is valid if: 1) it is between one (1) and the
prime, p, exclusive; and 2) if modular exponentiation of the Element
by the group order, r, equals one (1). If either of these conditions
are not true the received Element is invalid.

2.8.5.2.2. Element Validation for ECC Groups

Validating a received ECC Element involves: 1) checking whether the
two coordinates, x and y, are both greater than zero (0) and less
than the prime defining the underlying field; and 2) checking whether
the x- and y-coordinates satisfy the equation of the curve (that is,
that they produce a valid point on the curve that is not the point at
infinity). If either of these conditions are not met, the received
Element is invalid; otherwise, the Element is valid.

2.8.5.3. EAP-pwd-Confirm Exchange

The server computes Confirm_S according to Section 2.8.4, constructs
an EAP-pwd-Confirm/Request according to Section 3.2.3, and sends it
to the peer.

Upon receipt of an EAP-pwd-Confirm/Request, the peer validates the
length of the entire payload based upon the expected length of
Confirm_S (whose length is fixed by the agreed-upon random function).
If the length is incorrect, the peer MUST terminate the exchange and
free up any state allocated. If the length is correct, the peer
verifies that Confirm_S is the value it expects based on the value of
kp. If the value of Confirm_S is incorrect, the peer MUST terminate
the exchange and free up any state allocated. If the value of
Confirm_S is correct, the peer computes Confirm_P, constructs an EAP-
pwd-Confirm/Response according to Section 3.2.3, and sends it off to
the server. The peer then computes MK (according to Section 2.8.4)
and the MSK and EMSK (according to Section 2.9) and stores these keys

in state allocated for this exchange. The peer SHOULD export the MSK
and EMSK at this time in anticipation of a secure association
protocol by the lower layer to create session keys. Alternatively,
the peer can wait until an EAP-Success message from the server before
exporting the MSK and EMSK.

Upon receipt of an EAP-pwd-Confirm/Response, the server validates the
length of the entire payload based upon the expected length of
Confirm_P (whose length is fixed by the agreed-upon random function).
If the length is incorrect, the server MUST respond with an EAP-
Failure message, and it MUST terminate the exchange and free up any
state allocated. If the length is correct, the server verifies that
Confirm_P is the value it expects based on the value of ks. If the
value of Confirm_P is incorrect, the server MUST respond with an EAP-
Failure message. If the value of Confirm_P is correct, the server
computes MK (according to Section 2.8.4) and the MSK and EMSK
(according to Section 2.9). It exports the MSK and EMSK and responds
with an EAP-Success message. The server SHOULD free up state
allocated for this exchange.

2.9. Management of EAP-pwd Keys

[RFC5247] recommends each EAP method define how to construct a
Method-ID and Session-ID to identify a particular EAP session between
a peer and server. This information is constructed thusly:

    Method-ID = H(Ciphersuite | Scalar_P | Scalar_S)

    Session-ID = Type-Code | Method-ID

where Ciphersuite, Scalar_P, and Scalar_S are from the specific
exchange being identified; H is the random function specified in the
Ciphersuite; and, Type-Code is the code assigned for EAP-pwd, 52,
represented as a single octet.

The authenticated key exchange of EAP-pwd generates a shared and
authenticated key, MK. The size of MK is dependent on the random
function, H, asserted in the Ciphersuite. EAP-pwd must export two
512-bit keys, MSK and EMSK. Regardless of the value of len(MK),
implementations MUST invoke the KDF defined in Section 2.5 to
construct the MSK and EMSK. The MSK and EMSK are derived thusly:

    MSK | EMSK = KDF(MK, Session-ID, 1024)

[RFC4962] mentions the importance of naming keys, particularly when
key caching is being used. To facilitate such an important
optimization, names are assigned thusly:

o   EMSK-name = Session-ID | 'E' | 'M'| 'S' | 'K'

o   MSK-name = Session-ID | 'M'| 'S' | 'K'

where 'E' is a single octet of value 0x45, 'M' is a single octet of value 0x4d, 'S' is a single octet of value 0x53, and 'K' is a single octet of value 0x4b.

This naming scheme allows for key-management applications to quickly and accurately identify keys for a particular session or all keys of a particular type.

2.10. Mandatory-to-Implement Parameters

For the purposes of interoperability, compliant EAP-pwd implementations SHALL support the following parameters:

o   Diffie-Hellman Group: group 19 defined in [RFC5114]

o   Random Function: defined in Section 2.4

o   PRF: HMAC-SHA256 defined in [RFC4634]

o   Password Pre-Processing: none

3. Packet Formats

3.1. EAP-pwd Header

The EAP-pwd header has the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |L|M| PWD-Exch  |          Total-Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: EAP-pwd Header

Code

   Either 1 (for Request) or 2 (for Response); see [RFC3748].

Identifier

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

Length

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

Type

   52 - EAP-pwd

L and M bits

   The L bit (Length included) is set to indicate the presence of the
   two-octet Total-Length field, and MUST be set for the first
   fragment of a fragmented EAP-pwd message or set of messages.

   The M bit (more fragments) is set on all but the last fragment.

PWD-Exch

   The PWD-Exch field identifies the type of EAP-pwd payload
   encapsulated in the Data field. This document defines the
   following values for the PWD-Exch field:

   *   0x00 : Reserved

   *   0x01 : EAP-pwd-ID exchange

   *   0x02 : EAP-pwd-Commit exchange

   *   0x03 : EAP-pwd-Confirm exchange

   All other values of the PWD-Exch field are unassigned.

Total-Length

   The Total-Length field is two octets in length, and is present
   only if the L bit is set. This field provides the total length of
   the EAP-pwd message or set of messages that is being fragmented.

3.2. EAP-pwd Payloads

   EAP-pwd payloads all contain the EAP-pwd header and encoded
   information. Encoded information is comprised of sequences of data.
   Payloads in the EAP-pwd-ID exchange also include a ciphersuite
   statement indicating what finite cyclic group to use, what
   cryptographic primitive to use for H, and what PRF to use for
   deriving keys.

3.2.1. EAP-pwd-ID

   The Group Description, Random Function, and PRF together, and in that
   order, comprise the Ciphersuite included in the calculation of the
   peer's and server's confirm messages.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Group Description         | Random Func'n |     PRF     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Token                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Prep      |              Identity...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
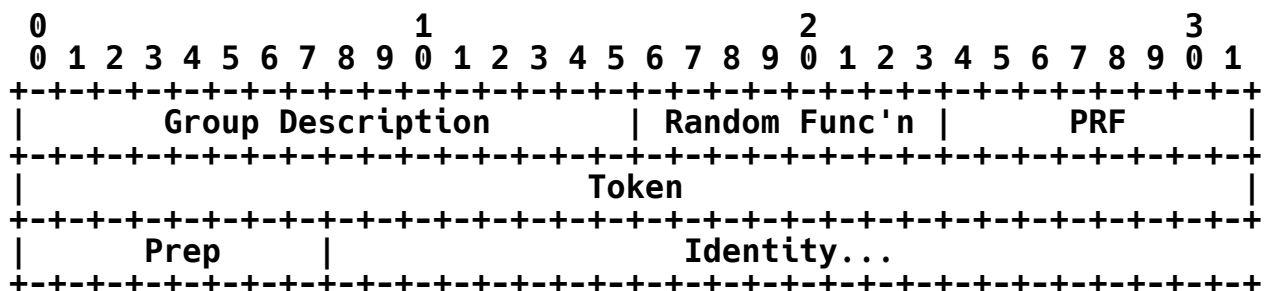
                    Figure 6: EAP-pwd-ID Payload

   The Group Description field value is taken from the IANA registry for
   "Group Description" created by IKE [RFC2409].

   This document defines the following value for the Random Function
   field:

   o   0x01 : Function defined in this memo in Section 2.4

   The value 0x00 is reserved for private use between mutually
   consenting parties. All other values of the Random Function field
   are unassigned.

   The PRF field has the following value:

   o   0x01 : HMAC-SHA256 [RFC4634]

   The value 0x00 is reserved for private use between mutually
   consenting parties. All other values of the PRF field are
   unassigned.

The Token field contains an unpredictable value assigned by the
server in an EAP-pwd-ID/Request and acknowledged by the peer in an
EAP-pwd-ID/Response (see Section 2.8.5).

The Prep field represents the password pre-processing technique (see
Section 2.7.2) to be used by the client prior to generating the
password seed (see Section 2.8.3). This document defines the
following values for the Prep field:

o   0x00 : None

o   0x01 : RFC2759

o   0x02 : SASLprep

All other values of the Prep field are unassigned.

The Identity field depends on the tuple of PWD-Exch/Code.

o   EAP-pwd-ID/Request : Server_ID

o   EAP-pwd-ID/Response : Peer_ID

The length of the identity is computed from the Length field in the
EAP header.

3.2.2. EAP-pwd-Commit

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                            Element                            ~
   |                                                               |
   ~                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                               |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               ~
   |                                                               |
   ~                  Scalar                 +-+-+-+-+-+-+-+-+-+-+-+
   |                                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

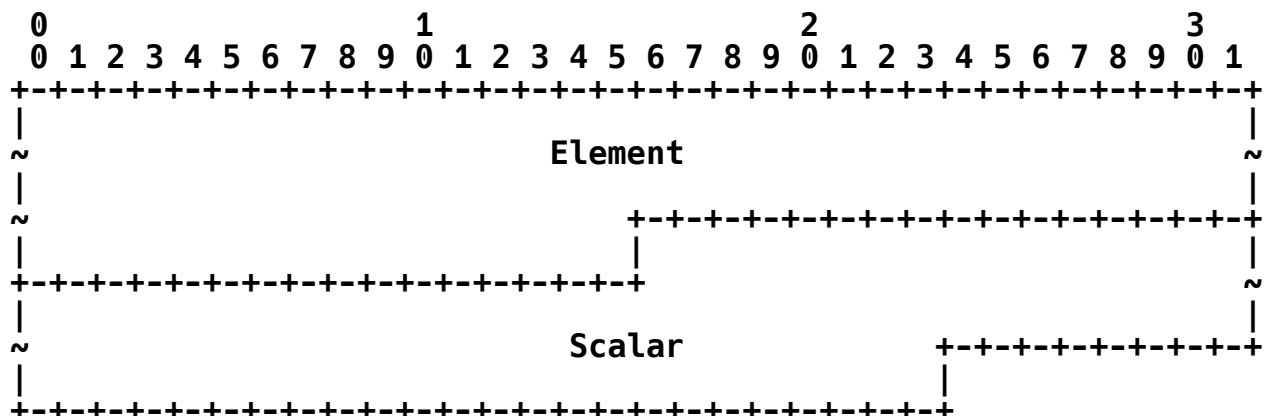   Figure 7: EAP-pwd-Commit Payload

The Element and Scalar fields depend on the tuple of PWD-Exch/Code.

o   EAP-pwd-Commit/Request : Element_S, Scalar_S

o   EAP-pwd-Commit/Response : Element_P, Scalar_P

The Element is encoded according to Section 3.3. The length of the
Element is inferred by the finite cyclic group from the agreed-upon
Ciphersuite. The length of the scalar can then be computed from the
Length in the EAP header.

3.2.3. EAP-pwd-Confirm

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                            Confirm                            ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
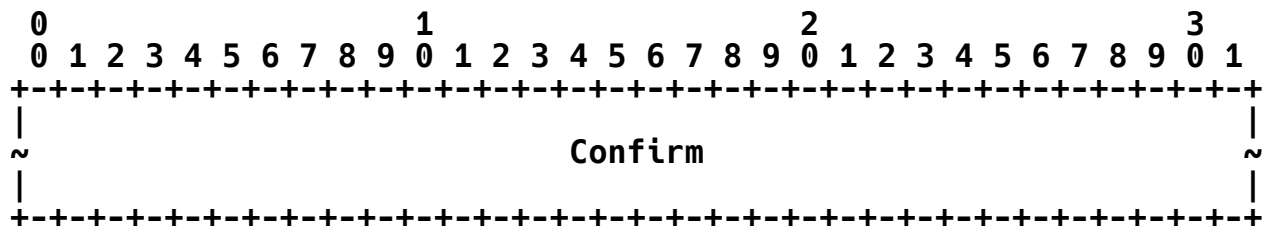
Figure 8: EAP-pwd-Confirm Payload

The Confirm field depends on the tuple of PWD-Exch/Code.

o   EAP-pwd-Confirm/Request : Confirm_S

o   EAP-pwd-Confirm/Response : Confirm_P

The length of the Confirm field computed from the Length in the EAP
header.

3.3. Representation of Group Elements and Scalars

Payloads in the EAP-pwd-Commit exchange contain elements from the
agreed-upon finite cyclic cryptographic group (either an FCC group or
an ECC group). To ensure interoperability, field elements and
scalars MUST be represented in payloads in accordance with the
requirements described below.

3.3.1. Elements in FFC Groups

Elements in an FFC group MUST be represented (in binary form) as
unsigned integers that are strictly less than the prime, p, from the
group's domain parameter set. The binary representation of each
group element MUST have a bit length equal to the bit length of the

binary representation of p. This length requirement is enforced, if
necessary, by prepending the binary representation of the integer
with zeros until the required length is achieved.

3.3.2. Elements in ECC Groups

Elements in an ECC group are points on the agreed-upon elliptic
curve. Each such element MUST be represented by the concatenation of
two components, an x-coordinate and a y-coordinate.

Each of the two components, the x-coordinate and the y-coordinate,
MUST be represented (in binary form) as an unsigned integer that is
strictly less than the prime, p, from the group's domain parameter
set. The binary representation of each component MUST have a bit
length equal to the bit length of the binary representation of p.
This length requirement is enforced, if necessary, by prepending the
binary representation of the integer with zeros until the required
length is achieved.

Since the field element is represented in a payload by the
x-coordinate followed by the y-coordinate, it follows that the length
of the element in the payload MUST be twice the bit length of p. In
other words, "compressed representation" is not used.

3.3.3. Scalars

Scalars MUST be represented (in binary form) as unsigned integers
that are strictly less than r, the order of the generator of the
agreed-upon cryptographic group. The binary representation of each
scalar MUST have a bit length equal to the bit length of the binary
representation of r. This requirement is enforced, if necessary, by
prepending the binary representation of the integer with zeros until
the required length is achieved.

4. Fragmentation

EAP [RFC3748] is a request-response protocol. The server sends
requests and the peer responds. These request and response messages
are assumed to be limited to at most 1020 bytes. Messages in EAP-pwd
can be larger than 1020 bytes and therefore require support for
fragmentation and reassembly.

Implementations MUST establish a fragmentation threshold that
indicates the maximum size of an EAP-pwd payload. When an
implementation knows the maximum transmission unit (MTU) of its lower
layer, it SHOULD calculate the fragmentation threshold from that
value. In lieu of knowledge of the lower layer's MTU, the
fragmentation threshold MUST be set to 1020 bytes.

Since EAP is a simple ACK-NAK protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field as is provided in IPv4.

EAP-pwd fragmentation support is provided through the addition of flags within the EAP-Response and EAP-Request packets, as well as a Total-Length field of two octets. Flags include the Length included (L) and More fragments (M) bits. The L flag is set to indicate the presence of the two-octet Total-Length field, and MUST be set for the first fragment of a fragmented EAP-pwd message or set of messages. The M flag is set on all but the last fragment. The Total-Length field is two octets, and provides the total length of the EAP-pwd message or set of messages that is being fragmented; this simplifies buffer allocation.

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

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

5. IANA Considerations

This memo contains new numberspaces to be managed by IANA. The policies used to allocate numbers are described in [RFC5226]. IANA has allocated a new EAP method type for EAP-pwd (52).

IANA has created new registries for PWD-Exch messages, random functions, PRFs, and password pre-processing methods and has added the message numbers, random function, PRF, and pre-processing methods specified in this memo to those registries, respectively.

The following is the initial PWD-Exch message registry layout:

o   0x00 : Reserved

o   0x01 : EAP-pwd-ID exchange

o   0x02 : EAP-pwd-Commit exchange

o   0x03 : EAP-pwd-Confirm exchange

The PWD-Exch field is 6 bits long. The value 0x00 is reserved. All
other values are available through assignment by IANA. IANA is
instructed to assign values based on "IETF Review" (see [RFC5226]).

The following is the initial Random Function registry layout:

o   0x00 : Private Use

o   0x01 : Function defined in this memo, Section 2.4

The Random Function field is 8 bits long. The value 0x00 is for
Private Use between mutually consenting parties. All other values
are available through assignment by IANA. IANA is instructed to
assign values based on "Specification Required" (see [RFC5226]). The
Designated Expert performing the necessary review MUST ensure the
random function has been cryptographically vetted.

The following is the initial PRF registry layout:

o   0x00 : Private Use

o   0x01 : HMAC-SHA256 as defined in [RFC4634]

The PRF field is 8 bits long. The value 0x00 is for Private Use
between mutually consenting parties. All other values are available
through assignment by IANA. IANA is instructed to assign values
based on "IETF Review" (see [RFC5226]).

The following is the initial layout for the password pre-processing
method registry:

o   0x00 : None

o   0x01 : RFC2759

o   0x02 : SASLprep

The Prep field is 8 bits long, and all other values are available
through assignment by IANA. IANA is instructed to assign values
based on "Specification Required" (see [RFC5226]).

6. Security Considerations

In Section 1.3, several security properties were presented that
motivated the design of this protocol. This section will address how
well they are met.

6.1. Resistance to Passive Attack

A passive attacker will see Scalar_P, Element_P, Scalar_S, and
Element_S. She can guess at passwords to compute the password
element but will not know s_rand or p_rand and therefore will not be
able to compute MK.

The secret random value of the peer (server) is effectively hidden by
adding p_mask (s_mask) to p_rand (s_rand) modulo the order of the
group. If the order is "r", then there are approximately "r"
distinct pairs of numbers that will sum to the value Scalar_P
(Scalar_S). Attempting to guess the particular pair is just as
difficult as guessing the secret random value p_rand (s_rand), the
probability of a guess is $1/(r - i)$ after "i" guesses. For a large
value of r, this exhaustive search technique is computationally
infeasible. An attacker would do better by determining the discrete
logarithm of Element_P (Element_S) using an algorithm like the baby-
step giant-step algorithm (see [APPCRY]), which runs on the order of
the square root of r group operations (e.g., a group with order 2^{160}
would require 2^{80} exponentiations or point multiplications). Based
on the assumptions made on the finite cyclic group in Section 2.3,
that is also computationally infeasible.

6.2. Resistance to Active Attack

An active attacker can launch her attack after an honest server has
sent EAP-pwd-Commit/Request to an honest peer. This would result in
the peer sending EAP-pwd-Commit/Response. In this case, the active
attack has been reduced to that of a passive attacker since p_rand
and s_rand will remain unknown. The active attacker could forge a
value of Confirm_P (Confirm_S) and send it to the EAP server (EAP
peer) in the hope that it will be accepted, but due to the
assumptions on H made in Section 2.3, that is computationally
infeasible.

The active attacker can launch her attack by forging EAP-pwd-Commit/
Request and sending it to the peer. This will result in the peer
responding with EAP-pwd-Commit/Response. The attacker can then

attempt to compute ks, but since she doesn't know the password, this is infeasible. It can be shown that an attack by forging an EAP-pwd-Commit/Response is an identical attack with equal infeasibility.

6.3. Resistance to Dictionary Attack

An active attacker can wait until an honest server sends EAP-pwd-Commit/Request and then forge EAP-pwd-Commit/Response and send it to the server. The server will respond with EAP-pwd-Confirm/Request. Now the attacker can attempt to launch a dictionary attack. She can guess at potential passwords, compute the password element, and compute kp using her p_rand, Scalar_S, and Element_S from the EAP-pwd-Commit/Request and the candidate password element from her guess. She will know if her guess is correct when she is able to verify Confirm_S in EAP-pwd-Confirm/Request.

But the attacker committed to a password guess with her forged EAP-pwd-Commit/Response when she computed Element_P. That value was used by the server in his computation of ks that was used when he constructed Confirm_S in EAP-pwd-Confirm/Request. Any guess of the password that differs from the one used in the forged EAP-pwd-Commit/Response could not be verified as correct since the attacker has no way of knowing whether it is correct. She is able to make one guess and one guess only per attack. This means that any advantage she can gain -- guess a password, if it fails exclude it from the pool of possible passwords and try again -- is solely through interaction with an honest protocol peer.

The attacker can commit to the guess with the forged EAP-pwd-Commit/Response and then run through the dictionary, computing the password element and ks using her forged Scalar_P and Element_P. She will know she is correct if she can compute the same value for Confirm_S that the server produced in EAP-pwd-Confirm/Request. But this requires the attacker to know s_rand, which we noted above was not possible.

The password element PWE/pwe is chosen using a method described in Section 2.8.3. Since this is an element in the group, there exists a scalar value, q, such that:

    PWE = q * G, for an ECC group

    pwe = g^q mod p, for an FFC group

Knowledge of q can be used to launch a dictionary attack. For the sake of brevity, the attack will be demonstrated assuming an ECC group. The attack works thusly:

The attacker waits until an honest server sends an EAP-pwd-Commit/
Request. The attacker then generates a random Scalar_P and a random
p_mask and computes Element_P = p_mask * G. The attacker sends the
bogus Scalar_P and Element_P to the server and obtains Confirm_S in
return. Note that the server is unable to detect that Element_P was
calculated incorrectly.

The attacker now knows that:

$$KS = (Scalar\_P * q + p\_mask) * s\_rand * G$$

and

$$s\_rand * G = Scalar\_P * G - ((1/q) \bmod r * -Element\_P)$$

Since Scalar_P, p_mask, G, and Element_P are all known, the attacker
can run through the dictionary, make a password guess, compute PWE
using the technique in Section 2.8.3, determine q, and then use the
equations above to compute KS and see if it can verify Confirm_S. But
to determine q for a candidate PWE, the attacker needs to perform a
discrete logarithm that was assumed to be computationally infeasible
in Section 2.3. Therefore, this attack is also infeasible.

The best advantage an attacker can gain in a single active attack is
to determine whether a single guess at the password was correct.
Therefore, her advantage is solely through interaction and not
computation, which is the definition for resistance to dictionary
attack.

Resistance to dictionary attack means that the attacker must launch
an active attack to make a single guess at the password. If the size
of the dictionary from which the password was extracted was D, and
each password in the dictionary has an equal probability of being
chosen, then the probability of success after a single guess is 1/D.
After X guesses, and removal of failed guesses from the pool of
possible passwords, the probability becomes 1/(D-X). As X grows, so
does the probability of success. Therefore, it is possible for an
attacker to determine the password through repeated brute-force,
active, guessing attacks. This protocol does not presume to be
secure against this, and implementations SHOULD ensure the size of D
is sufficiently large to prevent this attack. Implementations SHOULD
also take countermeasures -- for instance, refusing authentication
attempts for a certain amount of time, after the number of failed
authentication attempts reaches a certain threshold. No such
threshold or amount of time is recommended in this memo.

6.4. Forward Secrecy

   The MSK and EMSK are extracted from MK, which is derived from doing
   group operations with s_rand, p_rand, and the password element. The
   peer and server choose random values with each run of the protocol.
   So even if an attacker is able to learn the password, she will not
   know the random values used by either the peer or server from an
   earlier run and will therefore be unable to determine MK, or the MSK
   or EMSK. This is the definition of Forward Secrecy.

6.5. Group Strength

   The strength of the shared secret, MK, derived in Section 2.8.4
   depends on the effort needed to solve the discrete logarithm problem
   in the chosen group. [RFC3766] has a good discussion on the strength
   estimates of symmetric keys derived from discrete logarithm
   cryptography.

   The mandatory-to-implement group defined in this memo is group 19, a
   group from [RFC5114] based on Elliptic Curve Cryptography (see
   Section 2.2.2) with a prime bit length of 256. This group was chosen
   because the current best estimate of a symmetric key derived using
   this group is 128 bits, which is the typical length of a key for the
   Advanced Encryption Standard ([FIPS-197]). While it is possible to
   obtain a equivalent measure of strength using a group based on Finite
   Field Cryptography (see Section 2.2.1), it would require a much
   larger prime and be more memory and compute intensive.

6.6. Random Functions

   The protocol described in this memo uses a function referred to as a
   "random oracle" (as defined in [RANDOR]). A significant amount of
   care must be taken to instantiate a random oracle out of handy
   cryptographic primitives. The random oracle used here is based on
   the notion of a "Randomness Extractor" from [RFC5869].

   This protocol can use any properly instantiated random oracle. To
   ensure that any new value for H will use a properly instantiated
   random oracle, IANA has been instructed (in Section 5) to only
   allocate values from the Random Function registry after being vetted
   by an expert.

   A few of the defined groups that can be used with this protocol have
   a security estimate (see Section 6.5) less than 128 bits, many do not
   though, and to prevent the random function from being the gating
   factor (or a target for attack), any new random function MUST map its
   input to a target of at least 128 bits and SHOULD map its input to a
   target of at least 256 bits.

7. Security Claims

   [RFC3748] requires that documents describing new EAP methods clearly
   articulate the security properties of the method. In addition, for
   use with wireless LANs, [RFC4017] mandates and recommends several of
   these. The claims are:

   a. mechanism: password.

   b. claims:

       *   mutual authentication: the peer and server both authenticate
           each other by proving possession of a shared password. This
           is REQUIRED by [RFC4017].

       *   forward secrecy: compromise of the password does not reveal
           the secret keys -- MK, MSK, or EMSK -- from earlier runs of
           the protocol.

       *   replay protection: an attacker is unable to replay messages
           from a previous exchange to either learn the password or a
           key derived by the exchange. Similarly the attacker is
           unable to induce either the peer or server to believe the
           exchange has successfully completed when it hasn't.
           Reflection attacks are foiled because the server ensures that
           the scalar and element supplied by the peer do not equal its
           own.

       *   key derivation: keys are derived by performing a group
           operation in a finite cyclic group (e.g., exponentiation)
           using secret data contributed by both the peer and server.
           An MSK and EMSK are derived from that shared secret. This is
           REQUIRED by [RFC4017]

       *   dictionary attack resistance: this protocol is resistant to
           dictionary attack because an attacker can only make one
           password guess per active attack. The advantage gained by an
           attacker is through interaction not through computation.
           This is REQUIRED by [RFC4017].

       *   session independence: this protocol is resistant to active
           and passive attack and does not enable compromise of
           subsequent or prior MSKs or EMSKs from either passive or
           active attack.

       *   Denial-of-Service Resistance: it is possible for an attacker
           to cause a server to allocate state and consume CPU cycles
           generating Scalar_S and Element_S. Such an attack is gated,

though, by the requirement that the attacker first obtain
connectivity through a lower-layer protocol (e.g. 802.11
authentication followed by 802.11 association, or 802.3
"link-up") and respond to two EAP messages --the EAP-ID/
Request and the EAP-pwd-ID/Request. The EAP-pwd-ID exchange
further includes an anti-clogging token that provides a level
of assurance to the server that the peer is, at least,
performing a rudimentary amount of processing and not merely
spraying packets. This prevents distributed denial-of-
service attacks and also requires the attacker to announce,
and commit to, a lower-layer identity, such as a MAC (Media
Access Control) address.

*   Man-in-the-Middle Attack Resistance: this exchange is
    resistant to active attack, which is a requirement for
    launching a man-in-the-middle attack. This is REQUIRED by
    [RFC4017].

*   shared state equivalence: upon completion of EAP-pwd, the
    peer and server both agree on MK, MSK, EMSK, Method-ID, and
    Session-ID. The peer has authenticated the server based on
    the Server-ID, and the server has authenticated the peer
    based on the Peer-ID. This is due to the fact that Peer-ID,
    Server-ID, and the shared password are all combined to make
    the password element, which must be shared between the peer
    and server for the exchange to complete. This is REQUIRED by
    [RFC4017].

*   fragmentation: this protocol defines a technique for
    fragmentation and reassembly in Section 4.

*   resistance to "Denning-Sacco" attack: learning keys
    distributed from an earlier run of the protocol, such as the
    MSK or EMSK, will not help an adversary learn the password.

c. key strength: the strength of the resulting key depends on the
    finite cyclic group chosen. See Section 6.5. This is REQUIRED
    by [RFC4017].

d. key hierarchy: MSKs and EMSKs are derived from the MK using the
    KDF defined in Section 2.5 as described in Section 2.8.4.

e. vulnerabilities (note that none of these are REQUIRED by
    [RFC4017]):

    *   protected ciphersuite negotiation: the ciphersuite offer made
        by the server is not protected from tampering by an active
        attacker. Downgrade attacks are prevented, though, since

this is not a "negotiation" with a list of acceptable
            ciphersuites. If a Ciphersuite was modified by an active
            attacker it would result in a failure to confirm the message
            sent by the other party, since the Ciphersuite is bound by
            each side into its confirm message, and the protocol would
            fail as a result.

   *   confidentiality: none of the messages sent in this protocol
        are encrypted.

   *   integrity protection: messages in the EAP-pwd-Commit exchange
        are not integrity protected.

   *   channel binding: this protocol does not enable the exchange
        of integrity-protected channel information that can be
        compared with values communicated via out-of-band mechanisms.

   *   fast reconnect: this protocol does not provide a fast-
        reconnect capability.

   *   cryptographic binding: this protocol is not a tunneled EAP
        method and therefore has no cryptographic information to
        bind.

   *   identity protection: the EAP-pwd-ID exchange is not
        protected. An attacker will see the server's identity in the
        EAP-pwd-ID/Request and see the peer's identity in EAP-pwd-ID/
        Response.

8. Acknowledgements

The authors would like to thank Scott Fluhrer for discovering the
"password as exponent" attack that was possible in the initial
version of this memo and for his very helpful suggestions on the
techniques for fixing the PWE/pwe to prevent it. The authors would
also like to thank Hideyuki Suzuki for his insight in discovering an
attack against a previous version of the underlying key exchange
protocol. Special thanks to Lily Chen for helpful discussions on
hashing into an elliptic curve and to Jin-Meng Ho for suggesting the
countermeasures to protect against a small sub-group attack. Rich
Davis suggested the defensive checks to Commit messages, and his
various comments greatly improved the quality of this memo and the
underlying key exchange on which it is based. Scott Kelly suggested
adding the anti-clogging token to the ID exchange to prevent
distributed denial-of-service attacks. Dorothy Stanley provided
valuable suggestions to improve the quality of this memo. The
fragmentation method used was taken from [RFC5216].

9. References

9.1. Normative References

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

[RFC2759]   Zorn, G., "Microsoft PPP CHAP Extensions, Version 2",
             RFC 2759, January 2000.

[RFC3454]   Hoffman, P. and M. Blanchet, "Preparation of
             Internationalized Strings ("stringprep")", RFC 3454,
             December 2002.

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

[RFC4013]   Zeilenga, K., "SASLprep: Stringprep Profile for User
             Names and Passwords", RFC 4013, February 2005.

[RFC4282]   Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The
             Network Access Identifier", RFC 4282, December 2005.

[RFC4634]   Eastlake, D. and T. Hansen, "US Secure Hash Algorithms
             (SHA and HMAC-SHA)", RFC 4634, July 2006.

[RFC5226]   Narten, T. and H. Alvestrand, "Guidelines for Writing an
             IANA Considerations Section in RFCs", BCP 26, RFC 5226,
             May 2008.

[SP800-108] Chen, L., "Recommendations for Key Derivation Using
             Pseudorandom Functions", NIST Special
             Publication 800-108, April 2008.

[SP800-56A] Barker, E., Johnson, D., and M. Smid, "Recommendations
             for Pair-Wise Key Establishment Schemes Using Discrete
             Logarithm Cryptography", NIST Special
             Publication 800-56A, March 2007.

9.2. Informative References

[APPCRY]   Menezes, A., van Oorshot, P., and S. Vanstone, "Handbook
             of Applied Cryptography", CRC Press Series on Discrete
             Mathematics and Its Applications, 1996.

   [BM92]   Bellovin, S. and M. Merritt, "Encrypted Key Exchange:
                Password-Based Protocols Secure Against Dictionary
                Attack", Proceedings of the IEEE Symposium on Security
                and Privacy, Oakland, 1992.

   [BM93]   Bellovin, S. and M. Merritt, "Augmented Encrypted Key
                Exchange: A Password-Based Protocol Secure against
                Dictionary Attacks and Password File Compromise",
                Proceedings of the 1st ACM Conference on Computer and
                Communication Security, ACM Press, 1993.

   [BMP00]   Boyko, V., MacKenzie, P., and S. Patel, "Provably Secure
                Password Authenticated Key Exchange Using Diffie-
                Hellman", Proceedings of Eurocrypt 2000, LNCS
                1807 Springer-Verlag, 2000.

   [FIPS-197]   National Institute of Standards and Technology, FIPS Pub
                197: Advanced Encryption Standard (AES), November 2001.

   [JAB96]   Jablon, D., "Strong Password-Only Authenticated Key
                Exchange", ACM SIGCOMM Computer Communication
                Review Volume 1, Issue 5, October 1996.

   [LUC97]   Lucks, S., "Open Key Exchange: How to Defeat Dictionary
                Attacks Without Encrypting Public Keys", Proceedings of
                the Security Protocols Workshop, LNCS 1361, Springer-
                Verlag, 1997.

   [RANDOR]   Bellare, M. and P. Rogaway, "Random Oracles are
                Practical: A Paradigm for Designing Efficient
                Protocols", Proceedings of the 1st ACM Conference on
                Computer and Communication Security, ACM Press, 1993.

   [RFC2409]   Harkins, D. and D. Carrel, "The Internet Key Exchange
                (IKE)", RFC 2409, November 1998.

   [RFC3766]   Orman, H. and P. Hoffman, "Determining Strengths For
                Public Keys Used For Exchanging Symmetric Keys", BCP 86,
                RFC 3766, April 2004.

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

   [RFC4086]   Eastlake, D., Schiller, J., and S. Crocker, "Randomness
                Requirements for Security", BCP 106, RFC 4086,
                June 2005.

   [RFC4962]   Housley, R. and B. Aboba, "Guidance for Authentication,
                  Authorization, and Accounting (AAA) Key Management",
                  BCP 132, RFC 4962, July 2007.

   [RFC5114]   Lepinski, M. and S. Kent, "Additional Diffie-Hellman
                  Groups for Use with IETF Standards", RFC 5114,
                  January 2008.

   [RFC5216]   Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS
                  Authentication Protocol", RFC 5216, March 2008.

   [RFC5247]   Aboba, B., Simon, D., and P. Eronen, "Extensible
                  Authentication Protocol (EAP) Key Management Framework",
                  RFC 5247, August 2008.

   [RFC5869]   Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-
                  Expand Key Derivation Function (HKDF)", RFC 5869,
                  May 2010.

Authors' Addresses

   Dan Harkins
   Aruba Networks
   1322 Crossman Avenue
   Sunnyvale, CA 94089-1113
   USA

   EMail: dharkins@arubanetworks.com

   Glen Zorn
   Network Zen
   1310 East Thomas Street
   #306
   Seattle, WA 98102
   USA

   Phone: +1 (206) 377-9035
   EMail: gwz@net-zen.net